# Evolutionary Approaches to Mobile Robot Systems

Olumuyiwa Ibikunle Ashiru

A thesis submitted in partial fulfilment of the requirements of De Montfort University for the degree of Doctor of Philosophy.

Department of Computer Science
De Montfort University
Leicester LE1 9BH.

December 1997

## ABSTRACT

This thesis presents an investigation into the applicability of evolutionary computing techniques to problems in the robotics domain. Particular attention is given to the techniques of genetic algorithms (GA), applied to mobile robot path planning and genetic programming (GP) applied to the use of communication within controllers of collaborative teams of mobile robots. The thesis identifies and demonstrates in greater depth some of the key issues affecting the development of robust evolutionary based path-planning systems and demonstrates possible ways for maximising performance resulting from these issues. It also verifies, and in some cases cast doubt on existing results and commonly held beliefs with respect to the evolution of communicating controllers, as well as extending the scope of work in the area of evolution of controllers in communication based environments. The thesis illustrates the GP as a capable tool in this area and in the general area of control system extension and fault-tolerance system development.

This work shows that the use of evolutionary based approaches in the development of robotic systems is a viable alternative to existing methods, offering as its strong points three key features. Firstly, the ability to produce a diversity of potential solutions for a task, some offering general characteristics while others specific. Secondly, the ability to determine decompositional levels of a task both at a functional and a communication control level. Thirdly, the ability to produce optimal or near optimal solutions appropriate to the circumstance or information content.

The key results from the development of the GA based path planner show that representation dictates the effectiveness of path generation, with flexibility in its structure being the main pre-requisite. It also shows that issues of robustness can be tackled through the application of methods, which apply a control mechanism allowing for the expansion and contraction of the length of paths when appropriate, or methods that offer improved environment pre-processing.

The application of the GP to the evolution of communication based controllers for teams of mobile robots, shows that the evolutionary process can effectively manipulate low and high level functional units. Further, it shows that it is possible for the evolutionary process to identify the most appropriate information content for a task and how best to use it. It also showed that communication can be used in various ways (overriding, questioning and controlling) and that appropriate communication topologies as well as rates of communication can be established. As wells as the aforementioned results an additional finding was the fact that the information contained within the communication did not necessarily have to be task specific, in order for the evolutionary process to make use of it (although this in some cases increased the rate at which it is incorporated).

# CONTENTS


**1 INTRODUCTION.** ..... 16
- **1.1** Introduction. ..... 16
- **1.2** A brief history of autonomous robots. ..... 17
- **1.3** Evolutionary methods. ..... 20
- **1.4** Overview of thesis. ..... 21

**2 THEORETICAL FRAMEWORK: GENETIC ALGORITHMS AND PROGRAMMING.** ..... 22
- **2.1** Introduction. ..... 22
- **2.2** Genetic algorithms and their theoretical framework. ..... 24
  - **2.2.1** Genetic algorithms. ..... 24
  - **2.2.2** Genetic algorithm theoretical framework. ..... 35
- **2.3** Genetic programming. ..... 41
  - **2.3.1** Genetic programming. ..... 41
  - **2.3.2** Genetic programming theoretical framework. ..... 51
- **2.4** Summary. ..... 53

**3 LITERATURE REVIEW.** ..... 54
- **3.1** Introduction. ..... 54
- **3.2** Path planning. ..... 54
  - **3.2.1** Potential fields. ..... 55
  - **3.2.2** Free space methods. ..... 60
  - **3.2.3** Evolutionary path planning. ..... 66
  - **3.2.4** Path planning review summary. ..... 67
- **3.3** Robot communication and co-ordination. ..... 68
  - **3.3.1** Communication and co-ordination. ..... 69
  - **3.3.2** Application of evolution to communication and co-ordination. ..... 73
  - **3.3.3** Robot communication and co-ordination review summary. 78
- **3.4** In summary. ..... 79

**4 TASK I: PATH PLANNING.** ..... 81
- **4.1** Introduction. ..... 81
- **4.2** Implementation of problem. ..... 82
  - **4.3.1** Preparatory experiments. ..... 82
  - **4.3.2** Environment representation. ..... 85
  - **4.3.3** Path representation. ..... 86
  - **4.3.4** Definition of path planning constraints. ..... 87
  - **4.3.5** Obstacles avoidance method. ..... 88
  - **4.3.6** Implementing the Genetic algorithm. ..... 90
- **4.3** Experiments and analysis. ..... 91
  - **4.3.1** Experiment I. ..... 91

4.3.1.1 Results from experiment I. ..... 93
4.3.1.2 Discussion of results from experiment I. ..... 102
4.3.2 Experiment II. ..... 106
4.3.2.1 Results from experiment II. ..... 109
4.3.2.2 Discussion of results from experiment II. 116
4.3.3 Experiment III. ..... 118
4.3.3.1 Results from experiment III. ..... 122
4.3.3.2 Discussion of results from experiment III. 126
4.3.4 Experiment IV. ..... 127
4.3.4.1 Results from experiment IV. ..... 128
4.3.4.2 Discussion of results from experiment IV. ..... 131
4.4 Summary. ..... 132

5 INTERPRETER AND SIMULATED ENVIRONMENT. ..... 134
5.1 Introduction. ..... 134
5.2 Preliminary experiments. ..... 134
5.3 Genetic programming engine. ..... 137
5.4 Robot set-up. ..... 138
5.5 Simulator and robot environment model. ..... 140
5.5.1 Interpreter. ..... 140
5.5.2 Simulated environment. ..... 142
5.5.3 Distributed evaluation. ..... 147
5.6 Summary. ..... 148

6 TASK II: MEETING UP OF MULTIPLE ROBOTS. ..... 149
6.1 Introduction. ..... 149
6.2 Task overview and implementation. ..... 149
6.2.1 Experimental parameters. ..... 150
6.2.2 Task overview. ..... 151
6.2.3 Task implementation. ..... 154
6.3 Functions and terminals. ..... 156
6.4 Experiment and analysis. ..... 159
6.4.1 Overview of evolved control types. ..... 160
6.4.2 Experiment I. ..... 165
6.4.2.1 Results of experiment I. ..... 166
6.4.2.2 Discussion of results from experiment I. ..... 176
6.4.3 Experiment II. ..... 178
6.4.3.1 Results of experiment II. ..... 178
6.4.3.2 Discussion of results from experiment II. ..... 186
6.4.4 Experiment III. ..... 188
6.4.4.1 Results of experiment III. ..... 189
6.4.4.2 Discussion of results from experiment III. ..... 200
6.5 Summary. ..... 203

7 TASK III: KEEPING AN ENVIRONMENT SECURED. ..... 206
7.1 Introduction. ..... 206
7.2 Task overview and implementation. ..... 206
7.2.1 Experimental parameters. ..... 207
7.2.2 Task overview. ..... 207
7.2.3 Task implementation. ..... 210
7.3 Functions and terminals. ..... 214
7.4 Experiments and analysis. ..... 216
7.4.1 Results. ..... 217
7.4.2 Discussion of results. ..... 225
7.5 Summary. ..... 231

8 TASK IV: CIRCUIT COMPLETION. ..... 234
8.1 Introduction. ..... 234
8.2 Task overview and implementation. ..... 234
8.2.1 Experimental parameters. ..... 235
8.2.2 Task overview. ..... 236
8.2.3 Task implementation. ..... 237
8.3 Functions and terminals. ..... 240
8.4 Experiments and analysis. ..... 242
8.4.1 Results. ..... 243
8.4.2 Discussion of results. ..... 258
8.5 Summary. ..... 265

9 CONCLUSION. ..... 267
9.1 Overview. ..... 267
9.2 Further work. ..... 270
9.3 In conclusion. ..... 270

REFERENCES. ..... 272

APPENDIX_A. ..... 287

APPENDIX_B. ..... 300

APPENDIX_C. ..... 374

# LIST OF TABLES


4.1 Percentage of population allocated to each constraint for reproduction. ..... 84
4.2 List of test attributes for empty environment test cases. ..... 89
4.3 List of test attributes for initial cluttered environment test cases. ..... 89
4.4 Mean time taken to find first collision-free path for various repair-based configurations of GA-planner in obstacle free test cases and their associated standard deviations. ..... 94
4.5 Mean amount of excessive path length in the best paths produced for various repair-based configurations of GA-planner in obstacle free test cases and their associated standard deviations. ..... 95
4.6 Mean amount of excessive direction changes in the best paths produced for various repair-based configurations of GA-planner in obstacle free test cases and their associated standard deviations. ..... 95
4.7 Mean time taken to find first collision-free path in initial obstacle filled test cases and their associated standard deviations. ..... 97
4.8 Mean amount of excessive path length in best paths produced in initial obstacle filled test cases and their associated standard deviations. ..... 97
4.9 Mean amount of excessive direction changes in best paths produced in initial obstacle filled test cases and their associated standard deviations. ..... 98
4.10 List of test attributes for main test suit of cluttered environment test cases. .... 105
4.11 Mean time taken for GA-planners based on various path representations to find first collision-free path in obstacle filled test environments and their associated standard deviations. ..... 109
4.12 Mean amount of excessive path length in best path found by GA-planners based on various path representations in obstacle filled test environments and their associated standard deviations. ..... 110
4.13 Mean amount of excessive direction changes in best path found by GA-planners based on various path representations in obstacle filled test environments and their associated standard deviations. ..... 111
4.14 Mean direction change density in best path found by GA-planners based on various path representations in obstacle filled test environments and their associated standard deviations. ..... 111
4.15 Mean time taken to find first collision-free path using various GA-planner improvement routes in obstacle filled test environments and their associated standard deviations. ..... 120
4.16 Mean amount of excessive path length in best path produced using various GA-planner improvement routes in obstacle filled test environments and their associated standard deviations. ..... 120
4.17 Mean direction change density in the best path produced using various GA-planner improvement routes in obstacle filled test environments and their associated standard deviations. ..... 121

5.1 Percentage of population allocated to each constraint for reproduction in communication experiments. ..... 135

5.2 The execution stages of three programs in a time sliced interpreter using quantum length of six cycles. ..... 143

6.1 Standard functions and terminals available to GP. ..... 156
6.2 Communication receive functions and visual operators available to the GP. . 157
6.3 Functions and terminal sets and the experiments they are used in. ..... 157
6.4 Communication dependency fitness table for figure 6.9. ..... 166
6.5 Communication dependency fitness table for figure 6.11. ..... 167
6.6 Communication dependency fitness table for figure 6.12. ..... 169
6.7 Communication dependency fitness table for figure A.6 appendix A. ..... 170
6.8 Communication dependency fitness table for figure 6.13. ..... 171
6.9 Communication dependency fitness table for figure 6.14. ..... 172
6.10 Communication dependency fitness table for figure 6.15. ..... 173
6.11 Summary table of performance of the evolved controllers presented in experiment I. ..... 175
6.12 Communication dependency fitness table for figure 6.16. ..... 178
6.13 Communication dependency fitness table for figure 6.17. ..... 178
6.14 Communication dependency fitness table for figure 6.18. ..... 179
6.15 Communication dependency fitness table for figure 6.19. ..... 180
6.16 Communication dependency fitness table for figure 6.20. ..... 181
6.17 Communication dependency fitness table for figure 6.21. ..... 182
6.18 Communication dependency fitness table for figure A.7 appendix A. ..... 182
6.19 Communication dependency fitness table for figure A.8 appendix A. ..... 183
6.20 Summary table of performance of the evolved controllers presented in experiment II. ..... 186
6.21 Communication dependency fitness table for figure A.9 appendix A. ..... 188
6.22 Communication dependency fitness table for figure A.10 appendix A. ..... 188
6.23 Communication dependency fitness table for figure A11 appendix A. ..... 188
6.24 Communication dependency fitness table for figure 6.22. ..... 189
6.25 Communication dependency fitness table for figure 6.23. ..... 190
6.26 Communication dependency fitness table for figure 6.24. ..... 190
6.27 Communication dependency fitness table for figure 6.25. ..... 192
6.28 Communication dependency fitness table for figure 6.26. ..... 193
6.29 Communication dependency fitness table for figure 6.27. ..... 193
6.30 Communication dependency fitness table for figure 6.28. ..... 194
6.31 Communication dependency fitness table for figure 6.29. ..... 194
6.32 Communication dependency fitness table for figure 6.30. ..... 195
6.33 Communication dependency fitness table for figure 6.31. ..... 195
6.34 Communication dependency fitness table for figure 6.32. ..... 196
6.35 Communication dependency fitness table for figure 6.33. ..... 198
6.36 Summary table of performance of the evolved controllers presented in experiment III. ..... 200

7.1 States available for use by robot. ..... 211
7.2 Available functions and terminals. ..... 214
7.3 Functions and terminals accessible to individual system blocks. ..... 215

7.4 Summary table of performance of the evolved controllers reported in this chapter. ..... 226

8.1 List of available functions and terminals and their uses. ..... 240
8.2 Summary table showing the performance of the evolved controllers reported in this chapter. ..... 259

# LIST OF FIGURES


2.1 An overview of the functioning of a genetic algorithm. ..... 23
2.2 The functioning of one-point crossover operator used by the genetic algorithm. ..... 24
2.3 The functioning of two-point crossover operator used by the genetic algorithm. ..... 24
2.4 The functioning of uniform-crossover operator used by the genetic algorithm. ..... 25
2.5 The functioning of uniform mutation operator used by the genetic algorithm. ..... 25
2.6 Various interpretations of a program used by a GP. ..... 39
2.7 Initial state of programs before GP-based one-point crossover is applied. .... 42
2.8 Resultant structure of programs after GP-based one-point crossover has been applied. ..... 43
2.9 Resultant structure of programs after GP-based one-point context-preserving crossover has been applied. ..... 44
2.10 Initial program structure before GP mutation process commenced. ..... 45
2.11 Mutation of a terminal into a sub-tree by GP mutate operator. ..... 46
2.12 Mutation of a sub-tree into a terminal using GP mutation. ..... 46
2.13 Mutation of a terminal into a terminal, using GP mutation. ..... 46

4.1 An example of the relative path representation used by the GA planner. ..... 83
4.2 Resulting appearance of two obstacles after directed flow avoidance colouring has been applied to them. ..... 86
4.3 Test case A1 with optimal path. ..... 91
4.4 Test case A2 with optimal path. ..... 91
4.5 Test case A3 with optimal path. ..... 92
4.6 Test case A4 with optimal path. ..... 92
4.7 Path produced in test A1. ..... 93
4.8 Path produced in test A2. ..... 93
4.9 Path produced in test A3. ..... 94
4.10 Path produced in test A4. ..... 94
4.11 Scenario 1, deceptive environment. Optimal solution for test B1. ..... 96
4.12 Scenario 2, cluttered environment. Optimal solution for test B2. ..... 96
4.13 Scenario 3, very cluttered environment. Optimal solution for test B3. ..... 97
4.14 Sample path produced in deceptive test environment. ..... 101
4.15 Sample path produced in cluttered environment. ..... 101
4.16 Sample path produced in a very cluttered environment. ..... 102
4.17 Path produced using Fixed based planner in test 1. ..... 108
4.18 Path produced using nixed based planner in test 1. ..... 108
4.19 Path produced using dynamic based planner in test 1. ..... 108
4.20 Path produced using Fixed based planner in test 6. ..... 108
4.21 Path produced using nFixed based planner in test 6. ..... 108
4.22 Path produced using nDynamic based planner in test 6. ..... 108

4.23 Typical failing path in test 4. ..... 109
4.24 Path produced using contour avoidance system in test 6. ..... 119
4.25 Path produced using constraint reformalisation in test 6. ..... 119
4.26 Path produced using contour avoidance system in test 4. ..... 119

5.1 Modelling of robot and associated properties. ..... 136
5.2 Example of program containing all possible block types. ..... 138
5.3 Code to ensure that a value between 0.0 and 1.0 is always used for heading. ..... 140
5.4 Interaction between simulator, interpreter and GP engine. ..... 141

6.1a General form of straight liner. ..... 159
6.1b General path of a straight liner. ..... 159
6.2a General form of a snaker. ..... 160
6.2b General path of a snaker. ..... 160
6.3 General form of a jerk angle snaker. ..... 161
6.4 Simple sensor dependent collision resolution strategy. ..... 162
6.5 Sensor and collision dependent reversal strategy. ..... 162
6.6 Multiple sensor dependent reversal. ..... 162
6.7 Sensor dependent random reversal on collision. ..... 162
6.8 Continuous reversal. ..... 163
6.9 Controller exhibiting detrimental use of communication. ..... 165
6.10 Indifferent application of communication. ..... 167
6.11 Controller exhibiting optimal placing and usage of communication. ..... 167
6.12 Controller misusing communicated information. ..... 169
6.13 Range dependent controller type featuring optimal usage of communicated information. ..... 171
6.14 Effective controller employing no collision resolution strategy. ..... 172
6.15 Controller demonstrating dual axis's closeness threshold. ..... 172
6.16 Controller featuring the use of explicit communication dependent behaviour trigger. ..... 177
6.17 Controller exhibiting subtle explicit communication dependent behaviour trigger. ..... 178
6.18 Improved version of controller in figure 6.24. ..... 179
6.19 Most efficient version of controller 24 evolved. ..... 180
6.20 Snaker based controller fully utilising communicated information. ..... 181
6.21 Another controller which full utilises all the available communicated information. ..... 182
6.22 Effective controller produced using displacement communication over a localised range. ..... 189
6.23 Localised communication using Gdx as part of co-ordination calculations. ..... 190
6.24 Simple controller exhibiting general behaviour of solutions using localised communication. ..... 190
6.25 Initial controller utilising no collision resolution strategy and direct co-ordination with communicated information. ..... 192

6.26 Controller exhibiting probabilistic communication based trigger. ..... 192
6.27 Controller employing collision resolution strategy and direct co-ordination using communicated information. ..... 193
6.28 Controller using GRdy for co-ordination. ..... 194
6.29 Controller illustrating that the precise contents of information transmitted is not really relevant. ..... 194
6.30 Controller demonstrating the main role of GRdy as part of reversal strategy. ..... 195
6.31 Controller exhibiting use of all communicated information for co-ordination. ..... 195
6.32 Controller showing the use of all communicated information in both co-ordination and reversal strategy. ..... 196
6.33 Best controller using communicated visual information. ..... 198

7.1 Test environment for task. ..... 210
7.2 Default controller for robot with capacity to be overridden. ..... 211
7.3 Communication dependent controller. ..... 217
7.4 Communication dependent controller employing communicated information to allow default program to be executed. ..... 218
7.5 Controller using communication dependent state for co-ordination. 219
7.6 Controller exhibiting knowledge dependent communication and state co-ordination. ..... 219
7.7 Controller employing knowledge to determining communication type. ..... 220
7.8 Controller showing state dependent communication with visual dependent co-ordination. ..... 221
7.9 Controller exhibiting knowledge dependent docking. ..... 221
7.10 Controller employing last minute override avoidance check. ..... 222
7.11 Controller exhibiting additional collision resolution behaviour. ..... 222
7.12 Highly effective controller overriding all possible default program actions. ..... 222
7.13 Controller exhibiting knowledge dependent participation. ..... 223
7.14 Large controller giving relatively good performance. ..... 223
7.15 Controller exhibiting knowledge dependent participation. ..... 224
7.16 Controller displaying knowledge dependent communication and docking. ... 224

8.1 Test environment 1. ..... 237
8.2 Test environment 2. ..... 237
8.3 Initial effective controller. ..... 244
8.4 Communication command controller using program length to improve performance. ..... 244
8.5 Communication command controller using sub-population communication to improve performance. ..... 245
8.6 Controller using varying sized communication sub-populations. ..... 245
8.7 Controller applying alternate strategy using slow down command. ..... 245
8.8 Controller using non-standard communication sequence. ..... 245
8.9 Controller using explicit individualized communication. ..... 247

8.10 Controller exhibiting heterogeneous functionality and individualized communication. ..... 248
8.11 Controller utilizing both group and individualized communication. ..... 248
8.12 Controller using group based communication to produce 1:1 communication. ..... 249
8.13 Controller applying different change track rates to robots. ..... 249
8.14 Controller using group communication to omit communication to specific individuals. ..... 250
8.15 Controller using multiple command combinations along with main strategy. ..... 250
8.16 Controller using information request and communication sub-populations to produce improved alternate strategy. ..... 252
8.17 Controller exhibiting detrimental use of information requests. ..... 253
8.18 Simple information request incorporated within a multiple command action strategy. ..... 253
8.19 Compact use of multiple information request with main strategy. ..... 254
8.20 Information request used to determine communication sequence as well as command action population. ..... 254
8.21 Controller featuring heterogeneous functionality with varying information requests. ..... 256
8.22 Compact controller using minimal information request for command action employed and exhibiting collision order dependence. ..... 256
8.23 High performing controller using 1:1 command communication and multiple group orientated information requests. ..... 256
8.24 High performing controller based on group orientated communication and utilizing multiple information requests. ..... 257
8.25 Controller utilizing individualized information requests to determine which of two possible strategies to apply. ..... 257

# LIST OF GRAPHS


6.1 A comparison of the number of solutions produced in a simple information environment which did or did not use the locally available information. ..... 168

6.2 A comparison of the number of solutions produced in a displacement information environment which did or did not use this locally available information. ..... 174

6.3 A comparison of the number of solutions produced in a displacement information environment which did or did not use the globally available information. ..... 184

6.4 Cumulated total of solutions produced when the use of communicated information is the only method available for the robots to be aware of each other and when the information content contains displacement information. ..... 191

6.5 Cumulated total of solutions produced when the use of communicated information is the only method available for the robots to be aware of each other and when the information content contains relative displacement information. ..... 197

7.1 A comparison between the number of controllers evolved which did and didn't use communicated information. ..... 229

7.2 A comparison of the number of controllers evolved which used either task dependent or independent. ..... 229

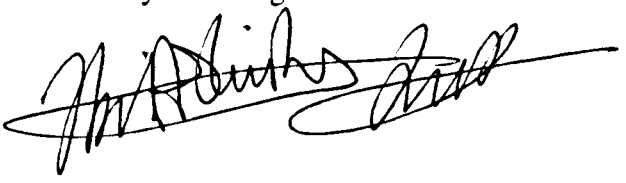
7.3 The change in the ratio of the best to mean fitness of the population of controllers over a single run. ..... 230

8.1 Tolerable solutions making effective use of communication. ..... 263

8.2 Typical change in the ratio of best to average fitness over a run. ..... 263

# DECLARATION

I hereby declare that this thesis is a record of work undertaken by myself and is not being submitted concurrently in candidature for any other degree.



O. I. Ashiru

# ACKNOWLEDGEMENTS

I would like to express my thanks to all the members of the intelligent and robotic systems research group, for the time they devoted to reading this thesis in all its drafts and for their comments and suggestions. I would also like to thank Peter Innocent for his time, insight and suggestions in regard to this work. Further I would like to express my sincerest thanks to my supervisors, Chris Czarnecki, Tom Routen, Roger Hutty and Paul Luker, for their advice, guidance and encouragement throughout the duration of this thesis, for without their belief in me it would never have come to fruition. Finally, thanks go out to Finnigan for his input but most importantly to John Ebohon for inspiring the desire and drive to achieve into those around him.

# Chapter 1

# INTRODUCTION

## 1.1 INTRODUCTION

A major attraction of industrial robots is their ability to perform activities effectively, repeatedly and consistently. This potential has been fully realised in static, well-structured environments such as factories [1,2]. However, the extension of robots to perform tasks in more natural everyday environments has been somewhat disappointing, which is partly due to the dynamic and unstructured nature of such environments. The application of autonomous controllers to mobile robots offers a solution to this problem and as a result adds a potentially wider range of potential applications to the repertoire of robots. The term autonomous implies a degree of awareness on the part of the subject, that is it instils the subject with an ability (innate or developable) to govern its own actions and be independent, which, when applied to robotics, implies the development of robots that are able to determine their own course of actions in all cases and not just blindly following programmed manoeuvres. The mobile nature of these robots allows the use of various forms of locomotion, for example driving, walking, diving or flying. These allow the development of robots capable of performing a broad range of tasks such as underwater exploration, loading and unloading goods, mail delivery, security guarding, visiting and probing distant planets, lunar mining to mention just a few. The varied nature of the application areas makes it difficult to pre-program or develop general-purpose programs for them. Instead, the autonomy has to be achieved by allowing them to adapt to their environment and the task at hand. Various such approaches have been proposed in the literature, but to date these have met with limited success.

The remainder of this chapter is structured as follows: in section 1.2 a brief review of

autonomous robots is presented; section 1.3 introduces evolutionary computational methods and section 1.4 presents an overview of the thesis.

## 1.2 A BRIEF HISTORY OF AUTONOMOUS ROBOTS

The classical approach to autonomous robots, which is based on traditional symbolic AI systems, has failed to provide suitable controllers owing to the inherent limitations of representations and the problem of scaleability. Such symbolic or knowledge-based approaches rely on hierarchical top-down functional decomposition and utilise centralised planner-based controllers. The decomposition of the problem into functional segments, which are then individually pursued, is based on the premise that by solving sub-problems and then plugging them back together, autonomous robots will be produced. This, however, has not been the case, since in many situations the sub-problems have proved to be just as intractable as a whole. Further, the use of planning-based controllers requires that an action space be searched in order to determine the next appropriate action. This leads to performance problems as the action space expands rapidly as problems become more complex. The use of a centralised model of the world has demonstrated a level of acceptable performance in the simplistic environments utilised in much research, but this has proved extremely restrictive in real-world situations, which are often dynamic and comprise of multiple robots. This shortcoming has been attributed to problems of consistency of world model, memory requirements, and distributed co-ordination and co-operation [2].

Brooks [3] argued against this traditional symbolic approach, pointing out its weaknesses and also disagreeing with its underlying concept of functional decomposition, in that decomposing a problem along the lines of the poorly understood field of human information processing is inappropriate and restrictive. As an alternative he proposed a behaviour based approach based on the subsumption architecture, in which the robots start out with the minimal set of competencies which are needed to ensure their immediate safety. Upon these basic abilities, higher level skills are added which subsume the lower

level ones, giving rise to a layered system, which exhibits incremental growth. These layers all run concurrently allowing for distributed functionality to emerge. The layers are hard-wired together by the designer. As a result of the interaction between the layers Brooks speculated that non-trivial high-level behaviour would emerge. This approach uses the real world as its own model so requires no internal modelling system. Although this subsumption approach overcomes many of the problems of the traditional AI approaches it also presents some of its own, for example:- the prediction of interaction of layers, wiring of layers and the arbitration between competing actions are just some of those experienced. These problems result in robots based on this approach to lack the real-world performance they exhibited in their test environments and raise again questions of scaleability.

An alternative behavioural approach to Brooks is that based on neural networks [4,5]. Here the problem is approached from the standpoint that intelligence does not just emerge, it is learned. The approaches maybe categorised into one of three learning schemes: supervised, unsupervised and reinforcement. The field of neural networks is still an on-going area of research but the main perceived benefits it offers are considered to be:

- generalisation capacities (allowing for adaptive behaviour),
- high resistance to noise and contradictions (leading to higher levels of robustness).

Such an outcome would obviate the need for the handcrafting of behaviours and allow for the emergence of complex behaviours built upon simpler ones. However some of the problems encountered in this area are:

- how to learn new strategies utilising previously acquired skills,
- how reward allocation for local actions which lead to global outcome is decided,
- how to deal with context awareness.

More recently, evolutionary based approaches have become an active area of research for the development of autonomous robots. Harvey et al [6] propose its use since it overcomes the complexity explosion associated with the subsumption approach and minimises the level of human pre-specification of solutions to the problem. This degree of independence of solution allows the evolutionary process to determine whether functional decomposition, behavioural decomposition, a hybrid or some other alternative is best for the task at hand. They also propose its use for adapting existing controllers to deal with changes in the environment. A similar approach is advocated by Mataric [7], but here higher level functional units are manipulated. This potential in the approach is also recognised by Brooks [8] in that he advocates switching from the standard approach to subsumption to the use of evolutionary defined behaviours (using incremental learning) and wirings, going so far as to propose a behaviour definition language for the task based on high level functional units. Mataric & Cliff [9] present an overview of some of the key areas of research, describing some of the main methods and highlighting areas to be addressed.

The work presented in this thesis focuses on this evolutionary based approach to robotics and applies it to the area of robot co-ordination. Firstly, it is concerned with co-ordination via the development of explicit planning strategies, secondly it addresses robot co-ordination using communication based strategies. The explicit planning aspect proposes applying a genetic algorithm to one of the basic axioms of robotics (path planning) as a way to overcoming existing problems in the field. The communication-based strategy addresses issues relating to the poorly researched field of communication and co-ordination in multiple robot environments. By investigating how evolutionary processes utilise communication in such environments (using the genetic programming paradigm), it is hoped progress can be made towards understanding the communication requirements of robot systems in general.

## 1.3 EVOLUTIONARY METHODS

The use of Darwinian inspired evolution as a search and optimisation tool has taken off in recent years (see Baeck & Schwefel [10] and Langdon & Qureshi [11]). Various areas are emerging and being investigated in what is being rapidly known as the field of evolutionary computation. These include techniques such as evolutionary design, genetic algorithms, evolutionary programming, genetic programming and classifier systems. The applications of these techniques are varied and range from parameter optimisation to product design and from machine learning to art. Their popularity can be attributed to their ability to search large search spaces efficiently and their independence from domain-specific knowledge. The two methods considered in this thesis are Genetic Algorithm (GA) and Genetic Programming (GP).

A genetic algorithm is an adaptive search procedure based on simplified models of evolution, which requires that the natural parameter set of the problem be coded as a finite length string over some finite alphabet. Simple genetic algorithms are composed of three basic operators, reproduction, crossover and mutation. Genetic programming is a variant of the GA, wherein, instead of fixed length strings coded over a finite alphabet, variable length, structured stings are used and a programming language replaces the alphabet. The population contains strings of candidate programs, each of which can be executed. Functionally equivalent crossover and mutation operators are used but with the additional role of ensuring only syntactically correct programs result from their application. The genetic algorithm was chosen for the task of path planning due to its innate ability to search large search spaces efficiently and the relative simplicity it offered in the implementation of both path representation and planner logic. The selection of genetic programming, for the investigation of communication based strategies, was due to the fact that it allowed for the problem to be encoded, genetically using more or less the natural terminology of the problem domain and that it offered greater flexibility for the manipulation and production of potential solutions.

## 1.4 OVERVIEW OF THESIS

This chapter has briefly reviewed some of the problems designers of autonomous robots face. Given the shortcomings of current approaches and the recognised potential of evolutionary computation, this thesis addresses the following question:

> *"Can evolutionary approaches applied to robotic problems help us to develop robust autonomous robots?"*

The question is addressed by considering two areas: firstly, the application of genetic algorithm to path planning and secondly, the use of genetic programming to produce control systems that utilise communication.

The format of the remainder of this thesis is as follows: chapter 2 presents the workings and theory behind the evolutionary methods employed in this thesis. Chapter 3 reviews the current literature with regard to path planning and the evolution of control systems in communication dependent environments. Chapter 4 presents a sequence of experiments aimed at developing a novel path planning technique based on genetic algorithms. Chapter 5 presents the environment and tools developed to investigate the use of the GP. Chapter 6 presents a multi-robot task to identify the ability of evolution to benefit from the presence of communication in its environment when evolving controllers. Chapter 7 extends this work using a different task to determine if controllers can be evolved which decide when, what and how often to communicate to ensure an appropriate level of co-operation in order to complete a task. Chapter 8 presents a task aimed at evolving controllers which use communication to directly control the behaviour of individual or entire groups of robots. Finally in chapter 9 conclusions are drawn about the applicability of the evolutionary paradigm to robot systems and recommendations for further work are made.

# Chapter 2

# THEORETICAL FRAMEWORK: GENETIC ALGORITHMS AND PROGRAMMING

## 2.1 INTRODUCTION

This chapter introduces the genetic algorithm and genetic programming. The aim is to familiarise the reader with both their operation and underlying theory.

The GA is a search and optimisation tool with inherent problem independence. Goldberg [12] identifies 3 classes of search techniques:

- Calculus based.
- Enumerative.
- Random.

The calculus based search methods can be further sub-divided into two more classes, direct and indirect. The direct class of methods finds the optimum of a search space by starting from a point on a function, that represents the space, and traversing it in the direction of a local gradient. Hill climbing is an example of this. Here the optimum is found by traversing the function following the steepest possible gradient. In contrast, the indirect class of search methods finds extrema by solving the resultant set of non-linear equations which result from setting the gradient of the objective function to zero. There are, however, two main drawbacks that affect the robustness of this class of methods, as regards their role as general search and optimisation techniques. The local nature of their searching, makes them highly susceptible to local minima wells, and the fact that they depend upon the existence of derivatives for their functioning, implying the existence of a high degree of smoothness in the search space. This is contrary to real world search spaces which are often noisy, contain discontinuities and are multimodal. These search

spaces cannot be tackled by calculus based methods unless some severe restrictions are placed on them.

Enumerative search methods are applied to finite or discretized search spaces. The search algorithm considers the objective function value of every point in the space, one at a time. The simplicity of these methods belies an inherent drawback, namely inefficiency of searching. Such methods' ability to produce timely results as the search spaces grows in either size, complexity or dimensionality is severely hampered. For this reason, their robustness as regards a general-purpose search and optimisation technique is also limited. An example of an enumerative search method is dynamic programming.

Random search methods such as random walks and schemes work by searching and saving the best results encountered. In the long run, however, these will do no better than enumerative methods states Goldberg [12]. However, randomised methods, ones which use knowledge of previous actions combined with random choice, as a tool for guiding highly exploratory search processes, offer a strong potential for the role of a generalised search and optimisation technique. Such an approach is the Genetic Algorithm.

The lack of robustness for the purpose of generalised searching and optimisation of conventional methods, notes Goldberg [12], does not mean they are, however, useless, just that they have a very restricted and specialised domain of application. The GA is proposed as a search and optimisation technique which can offer this general-purpose application.

Genetic Programming also falls into the randomised sub-class of methods, however it is not proposed as a generalised search method rather as a general method for automatic programming. Its approach differs significantly from the conventional methods of: machine learning, adaptive systems, automated logic, artificial intelligence, expert systems, and neural networks. Genetic programming represents the actual programming, of code. It is based upon a biologically inspired search mechanism (evolution) and it

requires no additional logic or knowledge about what is being programmed. It is not, however, the only evolutionary method for automatic programming, others such as the Classifier System and Beagle exist, but these process rules rather than traditional programming languages.

In the remainder of this chapter the functioning of both genetic algorithms and genetic programming will be presented along with their associated underlying theories. Section 2.2 will concentrate on genetic algorithms, section 2.3 on genetic programming and section 2.4 will present a summary of the chapter.

## 2.2 GENETIC ALGORITHMS AND THEIR THEORETICAL FRAMEWORK

This section introduces the functioning of a GA highlighting its key operators and the role they play. It also presents the theory underpinning their operation.

### 2.2.1 Genetic algorithms

GAs are an adaptive search and optimisation technique which employ the principles of natural selection as their guiding force. Goldberg [12] identified four characteristics which distinguish the GA from conventional search methods, which are that:

- they work with a coding of the parameter set, not the parameters themselves,
- they search from a population of points, not a single point,
- they use payoff (objective functions) information, not derivatives or other auxiliary knowledge,
- they use probabilistic transition rules, not deterministic rules.

For their adaptive search and optimisation procedure GAs apply simplified models of evolution, using the three main genetic operators: reproduction, crossover and mutation. GAs work on collections (termed populations) of potential solutions represented as

fixed length bit strings (termed chromosomes) which are initially generated randomly. The problem to be solved is specified by way of an objective function, which is used to evaluate each chromosome in turn (i.e. to determine the fitness of the solution represented by the chromosome). Using these fitness ratings another population of chromosomes (potential solutions) is generated biased in favour of the fittest members of the current population. By doing this repeatedly, each population on average improves upon the previous one hence directing the search in the direction of the optimal solution (see Figure 2.1). The termination criterion, is either the finding of the optimum or the performing of an arbitrary number of repetitions (generations).

The three genetic operators utilised by the GA are required for production of new populations of chromosomes. The functioning of these operators is given below.

Reproduction is a dual stage process by which chromosomes from the current population are selected to join the mating pool and then paired off. This selection is a random process which is biased in favour of those chromosome with better fitness ratings (roulette wheel selection). This gives those chromosomes with above average fitnesses a greater probability of contributing the information they contain to the next generation. The pairing of individuals within the mating pool is also done randomly but there is no bias applied.

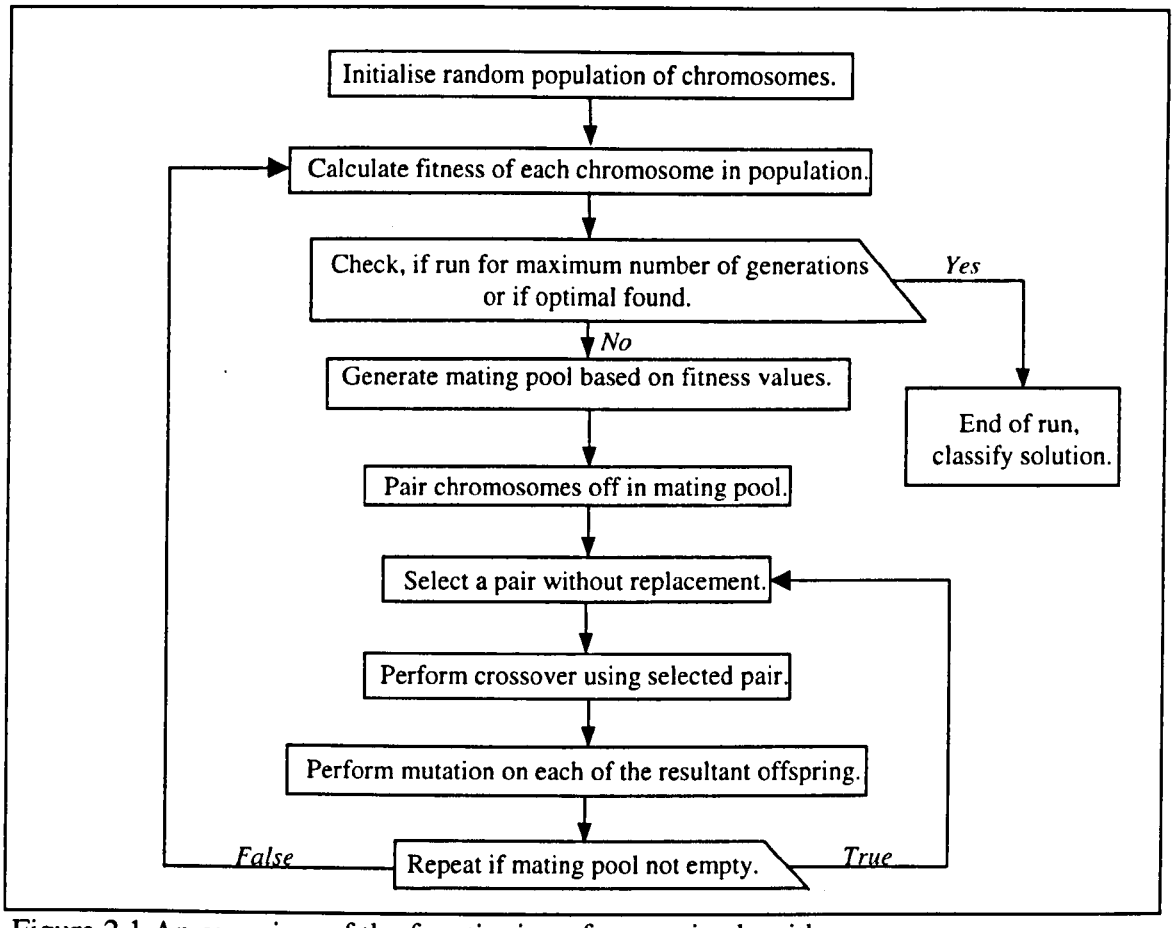


Figure 2.1 An overview of the functioning of a genetic algorithm.

Crossover is the process by which chromosomal information from the parents is exchanged to form offspring. The process acts on two parents and produces two offspring, which have similar properties to their parents (i.e. they inherit genetic characteristics of their parents). The mechanism by which information is exchanged in this process can take several forms: some of the most common are one-, two- point crossover and uniform-crossover. These three forms are covered below:

In one-point crossover a single common location within the chromosome length in both parents is chosen, then all the bits to the left of this point inclusively are copied to the relevant corresponding offspring (i.e. parent 1 to offspring 1 and parent 2 to offspring 2). The remaining bits are copied to the other offspring. An example of this can be seen in Figure 2.2.

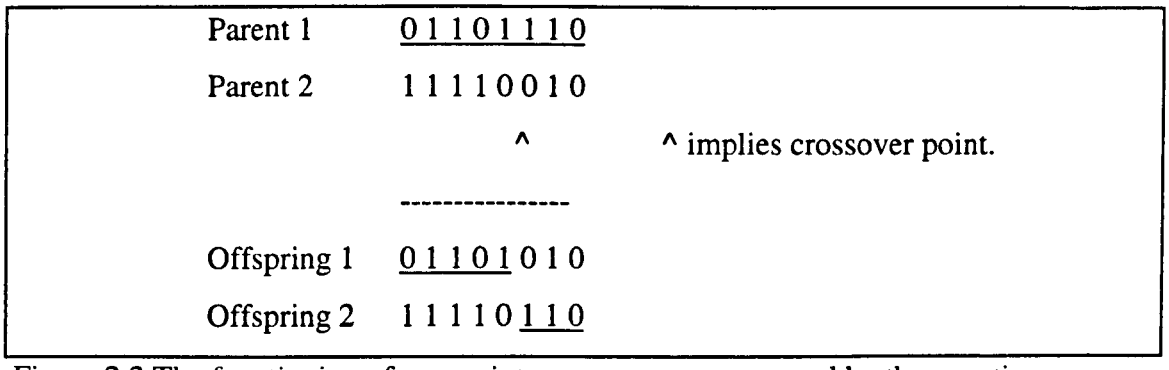



Figure 2.2 The functioning of one-point crossover operator used by the genetic algorithm.

In two-point crossover, two common points are chosen within the chromosome length of both the parents, such that the second point is closer to the right end of the chromosome than the first. All the bits outside these points are copied to the corresponding offspring and those inside to the other offspring. An example of this is shown in Figure 2.3.

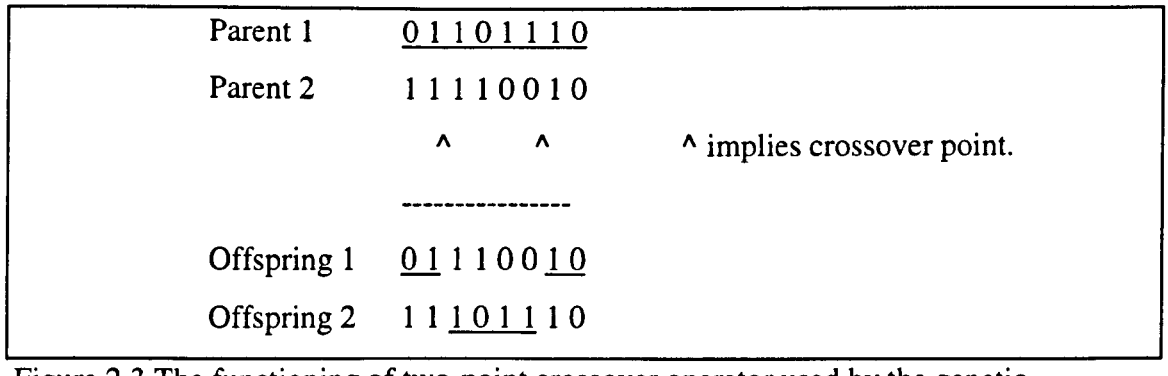



Figure 2.3 The functioning of two-point crossover operator used by the genetic algorithm.

In uniform-crossover all the corresponding bits in both parents are considered in turn. Starting from the leftmost bit position in each parent then moving sequentially through them, a random decision is made at each location as which parents contributes its current bit value to which offspring (only one parent can contribute per bit location to an offspring). An example of this is given in Figure 2.4.

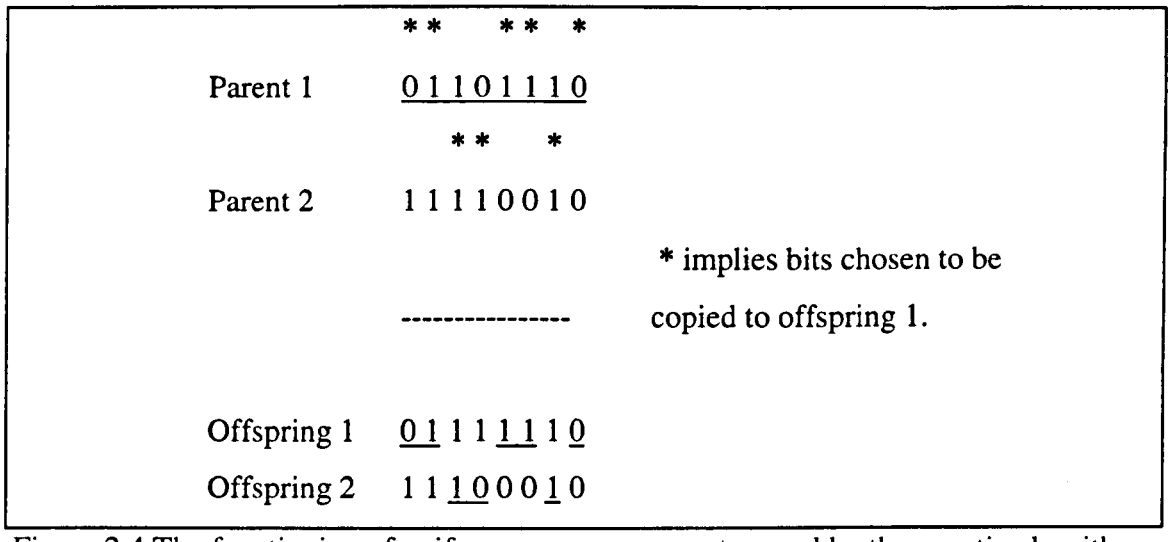


Figure 2.4 The functioning of uniform-crossover operator used by the genetic algorithm.

Mutation is the process by which errors in coping with data during the crossover stage is simulated. This has the effect of introducing diversity into the population as a whole. Uniform-mutation is the most common form of mutation, which works on the principle that there is a fixed probability of an error occurring during information exchange. So, as each bit of information is copied during the crossover stage, the likelihood of that bit being copied incorrectly gives rise to what is termed the mutation rate. Mutation here takes the form of flipping/inverting the bit value being copied. The mutation operator is usually applied after the crossover process is completed and is applied to the offspring's bit strings one at a time. Starting at the leftmost bit in the chromosome and traversing its length, the following sequence of instructions is applied at each location: generate a random number; if this number is less than the mutation rate, then flip the current bit otherwise leave it unchanged. The example in Figure 2.5 shows the outcome of applying this process to an offspring.

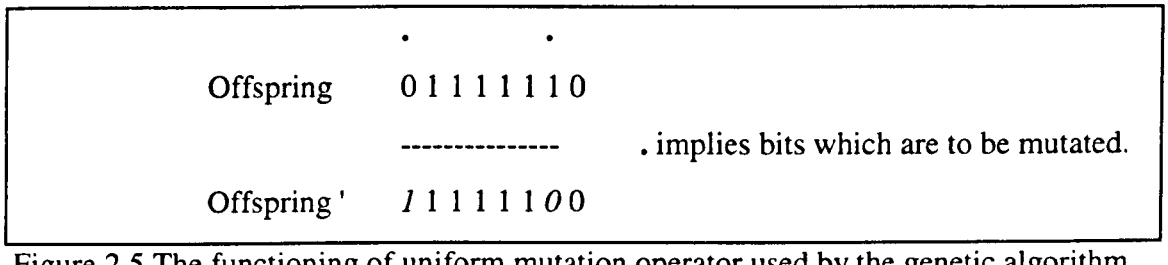


Figure 2.5 The functioning of uniform mutation operator used by the genetic algorithm.

In the implementation of both evolutionary methods used here, all those genetic operators which utilise random values have their own distinct random stream associated with them (i.e. each of these operators has its own distinct random number generator). This decision was taken so as to decouple the dependency between such operators, allowing the appropriate types of operators for a problem to be established in an independent manner. So, in the case of, say, crossover operators, three forms are available each of which consumes varying amounts of random values per execution. If a single stream were used, then changing the crossover operator would affect the sequence of random values any subsequent operators will receive. Therefore, any change in performance cannot be solely attributed to the new crossover operator, for it may also in part be attributable to a change in the decisions taken by other operators. However, the use of independent streams allows the crossover operator to be changed and any subsequent change in performance be attributable to that change. The use of multiple random streams requires for each stream to be independently initialised at the start of each run.

One of the problems suffered by many search techniques is convergence to a sub-optimal solution (termed here as premature convergence), which implies that the solution found is only a local optimum and not a global one. As a way of minimising this effect in the GA two methods are available, run the GA multiple times with different initial populations on each run (Koza [13]) or use some form of scaling operator to slow the rate at or degree to which highly fit chromosomes direct the searching process (Goldberg [12]).

The most common scaling operators are linear scaling and ranking, other methods that have been proposed but will not be discussed here are, duplicate elimination (Davis [14]) and incest prevention (Eshelman & Schaffer [15]).

Linear scaling is applied to all the fitness values of the chromosomes in the current population. It works by altering the fitness values of the chromosome such that they are all positive and range from 0 to n times the average fitness (where n is the scaling

factor). After applying this operator, the worst chromosome will have a fitness value of zero and the best n times the populations average fitness, all intermediary values have their values scaled linearly within this range.

Ranking also works on the fitness values of all the chromosomes of the current population, but in a slightly different way. It first uses the fitness of the chromosomes to order them (from worst to best), and then it reassigns a rank to the chromosomes instead of the fitness value. The rank indicates its position in the ordering of chromosomes and this starts from 0 and goes up in steps of r. Using this technique, no two chromosomes will ever have the same resultant fitness value. This is of use when or if the chromosome space consists of similar or equally fit individuals (usually towards the end of a run), since it adds a method for constantly grading these individuals.

So far, the implementation of this search method has remained true to its biological origins, but it is at the stage of efficiency where they start to diverge. Real-life biological systems have at their disposal a few million years to evolve to their optimum level. However, for the real-life application of a search technique there is always a finite amount of time usually measured in seconds or fractions of seconds. As a result, additional techniques not present in nature are often employed to speed up the search time. One such technique is constraining the search space of the GA. Since GAs operate by manipulating binary values, for a given number of bits there is a maximum value it can represent. If the range of values used is less than this maximum value then the GA will waste time processing solutions containing these additional values. By ensuring that the resultant of the crossover and mutation processes produce only values within the appropriate range, this wasted processing time can be avoided. Operators that perform this task are called filter/repair operators and tend to be problem specific. A side effect of this is to also ensure that only valid solutions are produced [16].

Another departure from that of nature apparent in the implementation of the GA is the alphabet used to construct the chromosomes. In nature, the alphabet consists of four letters {A, C, G, T} where as the most commonly applied alphabet in GAs consists of

only 2 letters {0, 1}. This is due to the facts that, GAs are implemented on computers whose base alphabet is binary and that binary codings have small cardinality, which helps reduce the search times when hunting for important similarities.

The simplest possible coding using this binary alphabet is to allocate a single bit in the chromosomes to each possible alternative (the most common is a group of bits corresponding to decimal values). Where '0' means off or not present and '1' means on or present. A simple example to illustrate this point follows:

A machine has 8 speeds, each of which can be combined; the problem is to discover which combinations of speeds give the best results.

For this problem, a chromosome length of 8 bits is required, where each bit will correspond to a possible speed, for example the string '01000110' would correspond to the use of speeds 7, 3 and 2 (if numbering from left to right in descending order). As mentioned early the most common coding mechanism is interpreting the bits as groups of numbers. Using this method, the previous binary sequence would be interpreted as 70.

These codings give a large degree of flexibility but they do not provide the variety of options required for tackling the myriad of problems presented in science, business and engineering. Subsequently Goldberg [12] states that coding GAs is somewhat of an art, but he highlights two guiding principles to help decide on the appropriate coding mechanism for a given problem. These are:

- The principle of meaningful building blocks. The user should select a coding so that short, low order schemata (see section 2.2.2) are relevant to the underlying problem and relatively unrelated to schemata over fixed positions.
- The principle of minimal alphabets. The user should select the smallest alphabet that permits natural expression of the problem.

Goldberg further states that the robustness of GAs is such that they are very forgiving if the wrong coding is chosen.

So far the coding for the chromosomes has only considered a single parameter/constraint, but many of the real-life problems consist of multiple parameters. These can be coded by concatenating multiple single parameter binary chromosomes together to form a single chromosome.

Goldberg [12] also proposes a collection of more advanced operators in addition to the main ones used by the GA, these are: dominance, diploidy and abeyance, inversion and other reordering operators, segregation and multiple chromosome structures, translocation, duplication and deletion, sexual determination and differentiation, niche and specification and mating restriction. Davis [14] proposes the use of uniform crossover in place of the standard one point suggested by Goldberg in [12]. This is because using one-point crossover, it would not be possible to preserve many of the possible short length schemata which may occur in the centre as well as in the left most bits of a chromosome. Davis commented that two-point crossover allowed for the preservation of more of these schemata than one point but still not as much as uniform crossover which allows for all the possible short defining length blocks to be preserved. However, the process it employs to preserve these blocks is very disruptive to the chromosome. Eshelman & Schaffer [15] concluded that the use of uniform-crossover with an elitist type of reproduction is the perfect combination. This is because with the traditional GA set-up the crossover process is responsible for both exploration and preservation of schemata. However, by employing elitist reproduction (where some of the best of the current population are copied unchanged into the next population) some or all of the burden of preservation is lifted from the shoulders of the crossover operator, allowing it to focus on exploration. The highly disruptive nature of the uniform crossover operator allows for a high degree of exploration. Eshelman and Schaffer state that one-point crossover and other such low-disruptive crossover operators are best applied in non-overlapping populations i.e. when the only way to preserve schemata is to pass them through offspring. Levenick [17] suggests the insertion of introns as a way

of improving the performance of a GA. An intron is a non-functional section of data, the purpose of which is to effectively decrease the disruptiveness of the crossover and mutation operators further.

Husbands and Mill [18] highlight the fact that by using co-evolving populations the implicit parallelism that exists within combinatorial problem optimisation is greatly exploited. Goldberg et al [19] introduce the concept of messy GA (mGA) highlighting the properties and proposing them as a viable field of research. Messy GAs differ from traditional GAs in four ways:

- they use variable length strings, which may be over or under specified,
- they use simple cut and splice operators in place of fixed length crossover operators,
- they divide the evolutionary process into two phases, primordial and juxtapositional,
- they have the ability to use competitive templates in order to accentuate salient building blocks.

The potential benefits they forecast for messy GAs is that they can avoid local minima traps in deceptive problems and they can solve such bounded deceptive problems to a global optima in a time that grows no more quickly than a polynomial function of the number decision variables on a serial machine or as a logarithmic function of the number of decision variables on a parallel machine.

The standard GA approach, however, is to optimise on only a single scalar value. In those problems that require multiple constraints, a solution to this would be to combine these constraints into some form of sum or weighted sum that could then be evaluated. However, this approach is somewhat restrictive since it does not allow any significant prioritisation method. The approach implemented here employs concepts proposed by Schaffer [20]. This technique retains the multi-dimensional nature of the fitness value and allows for varied forms of prioritisation of constraints. The constraint set is treated

as a mathematical vector. So under this scheme a given fitness vector $X$ is considered less than another fitness vector $Y$ (or non-dominated by) if all corresponding $X$ vector values are less than those of $Y$, and greater than (dominated by/ inferior to) $Y$, if all its corresponding vector values are greater than $Y$. An additional relationship is further employed, 'partially-less-than' which is defined as follows:

$X$ is considered partially-less-than $Y$ if at least one of $X$'s fitness vector values is less than its corresponding one in $Y$ and all others are equal to their corresponding vector value in $Y$.

Now, instead of choosing individuals for the mating pool based on one value, this is now done based on multiple values. Each constraint will contribute a certain percentage of the offspring for the next generation. Those offspring selected on a given constraint are paired off together, unless there is an odd number in the pool, in which case the final one is paired off with the first spare from the other constraints (it is assumed the population size is always even.) An example of applying the partially-less-than ($\nless$) operator is given below:

three fitness vectors [1,15,25], [2,20,25]; [1,5,25] are to be ordered using the partially-less than operator. This gives rise to the following ordering:

$$[1,5,25] \nless [1,15,25] \nless [2,20,25].$$

This vector method is analogous to having seven separate GAs each sharing the same chromosome pool but each with potentially different population sizes and each working on a single but different constraint.

Using this method the percentage of the mating pool attributed to each constraint will affect the performance of the GA, in that if an important constraint is given a small percentage then the rate at which it will traverse its search space will be very slow compared to the others. This will affect the rate at which the GA approaches the optimum.

Also if a less important constraint is given a high percentage then this will tend to steer the population towards a global sub-optimal value but local optimum for the constraint. The high order constraints as well as those which are easy to accomplish are given the lion's share of the reproduction resource. Those constraints, which are easy to accomplish, should not be over loaded, since population size is a finite commodity.

It can be seen from above that two forms of prioritisation are possible, firstly through the ordering of constraints in the vector and secondly through the allocation of processing resources to the constraints. Both these methods still, however, allow the GA to tackle the problem in a more systematic form.

### 2.2.2 Genetic algorithm theoretical framework

The schema theorem is the theory underpinning the functioning of GAs. It allows a scientific language to be defined upon which rigorous discussions and validations can be presented. The main premise of the schema theorem is based around the implicit parallel processing and preserving of the similarities that can exist within strings. Central to this is the rate at which beneficial and detrimental similarities grow and decay from generation to generation. The theorem requires two main tools: a way for expressing similarities in strings and a way for defining the properties of these similarities.

The schema theorem is concerned with similarities, so as such, the interest is no longer in strings as strings, rather in the elements that constitute them. In order to express these elements (i.e. similarities) within strings, schema theory utilises similarity templates or schema. This is defined as similarity template, which describes a subset of strings with similarities at certain string positions. Schema theory applies to genetic alphabets of all sizes, but here a binary alphabet will be considered. To produce schemata, the meta symbol '*' is added to the alphabet, giving in this case the extended alphabet {0,1,*}. This meta symbol means "do not care", the meta symbol is not actually explicitly processed by the GA it just allows for discussions of well-defined similarities among finite-length strings over finite alphabets. Schema themselves are pattern matchers and a

schema is consider to match another if at every location a 1 matches with a 1, a 0 matches with a 0 or a * is present. For example the schema '*100' matches with '1100' and '0100'. A given fixed-length string will always contain more schemata than possible produceable strings. That is the number of possible strings of length **l** from an alphabet of cardinality **k** is $k^l$, whereas the number of possible schemata in such a string is $(k+1)^l$. This implies their are more similarities in a population of strings than their are possible strings (i.e. $n*3^l$ versus $n*2^l$, where **n** is the population size). This is a beneficial feature, which will be covered later on in this section.

There are two main properties that help distinguish between schemata: these are, defining length and schema order, which allow for both the range and the specificity of schemata to be quantified. The defining length allows for the range of schemata to be stated. It is defined as the distance between the first and last specific (i.e. 1 or 0) string positions and is written as $\delta(\mathbf{H})$. Where **H** is the schema whose length is being defined.

E.g. if H='101*1**' then $\delta(H)=4$ and if H='1******' then $\delta(H)=0$.

The specificity of a schema is related to its order, which is defined as the number of fixed positions (i.e. 1 or 0) it contains, and is written as $\mathbf{o(H)}$. Where **H** is the schema whose order is to be determined.

E.g. if H='101*1**' then $o(H)=4$ and if H='1******' then $o(H)=1$.

Using these properties a rigorous discussion and classification of string similarities can take place.

Not all similarities (schemata) are equal, some are beneficial others detrimental, still others indifferent. The similarities in highly fit strings can guide the search in a positive direction, while the similarities in poorly fit strings can lead it astray. Therefore, what is needed is for those highly fit similarities to be encouraged and those poor or indifferent

similarities, discouraged. In the following sections, it will be shown that this is indeed what the schema theory identifies.

It was mentioned earlier that the GA processes $n*3^l$ schemata (where **n** is the population size) during a generation. The question is then, how many of these schemata are actually processed in a meaningful way, and how do they effect the growth and decay of important schemata between generations. These questions can be addressed by considering the effect the three main operators (reproduction, crossover and mutation) have on the schemata.

For reproduction, since fitter strings have higher probabilities of selection, on average they are therefore given an ever-increasing number of samples relative to the observed best similarity patterns. Golberg [12] expresses this as follows:

At time **t** there are **m** examples of a particular schema, say **H**, contained within a population **A**(t).

$$m \equiv m(H, t). \qquad \text{Equation 2.1}$$

This says there are possibly different quantities of different schemata **H** at different times **t**. Now, consider a string in population **A**, say $A_i$, (using non-overlapping populations with replacement) its probability of being selected is $P_i$. Where,

$$P_i = f_i / \Sigma f_j, \qquad \text{Equation 2.2}$$

where fi is the fitness of string Ai and $\Sigma f_j$ is the sum of all fitness values in the population. It is expected that at a time **t+1**, **m(H, t+1)** representations of schema **H** will be present in the current population, that is,

$$m(H, t+1) = m(H, t) * n * f(H) / \Sigma f_j. \qquad \text{Equation 2.3}$$

Where **n** is the population size, **f(H)** is the average fitness of the string representing schema **H** at time **t**. This is the reproduction growth equation. However, since the average fitness of the population $\bar{f}$ can be expressed as

$$\bar{f} = \Sigma f_j / n, \qquad \text{Equation 2.4}$$

using this, the reproduction growth equation can be re-written as,

$$m(H, t+1) = m(H, t) * f(H) / \bar{f}. \qquad \text{Equation 2.5}$$

Hence, it can be seen that a particular schema grows as a ratio of the average fitness of the schema to the average fitness of the population. Implying above average schemata get an increasing number of samples in subsequent generations and below average schemata get a decreasing number of samples. Additional information can be obtained from this growth equation about exactly what form this growth and decay takes. If it is assumed that a particular schema, **H**, remains above the average fitness by the amount (c * $\bar{f}$) (where c is constant), this allows the growth equation to be re-written as,

$$m(H, t+1) = m(H, t) * ( \bar{f} + (c * \bar{f})) / \bar{f} = m(H, t) * (1+c). \qquad \text{Equation 2.6}$$

Starting with t=0, the equation becomes,

$$m(H, t) = m(H, 0) * (1 + c)^t. \qquad \text{Equation 2.7}$$

This is effectively a geometric progression, implying an exponentially increasing number of samples for above average schema and an exponentially decreasing number of samples for below average schemata.

Crossover (assuming single-point crossover is used) may disrupt schema, however short defining length schemata are less likely to be disrupted or destroyed and be reproduced at a good sampling rate by reproduction. Goldberg [12] expressed this as follows.

The probability of a schema surviving the crossover process is varied. However, it is possible to give a lower bound of survival, $\mathbf{P_s}$. This can be expressed as

$$P_s = 1 - \delta(H) / (l-1), \qquad \text{Equation 2.8}$$

where **l** is the length of the string, schema **H** is in. However, if at a particular mating, crossover is itself applied in a probabilistic manner, at rate say $P_i$, then the crossover survival equation can be re-written as,

$$P_s >= 1-P_i * \delta(H) / (l-1).$$ Equation 2.9

From here, it can be seen that the smaller $\delta(\mathbf{H})$ is the better the survival chances are for schema **H**.

The combined effect of crossover and reproduction can be seen from the following equation (assuming independence of operation between these operators),

$$m(H, t+1) >= m(H, t) * (f(H) / \bar{f}) * [1 - P_i * (\delta(H) / (l-1))].$$ Equation 2.10

The effect of this combination is clear, schema H will grow and decay depending upon a multiplication factor. This factor is, whether or not the schema is above or below the population average and whether it has a relatively short or long defining length. So, above average short defining length schemata clearly have more samples.

Mutation at its usual low rates (Goldberg [12] suggests typical mutation rates of around 0.001) rarely disrupt the schemata. Goldberg [12], used the Equations 2.11 and 2.12 to show this.

First, the probability of an element in the schema surviving mutation must be determined. If the rate of mutation is $P_m$, then $(1-P_m)$ gives an element's survival chances. Now for the schema to survive the mutation process intact, all of its elements must survive mutation, assuming all mutation operations are independent of each other, then this can be expressed as,

$$(1-P_m)^{o(H)} \approx 1-o(H) * P_m, \text{ for small values of } P_m.$$ Equation 2.11

Finally, by combining the equations from all three operators their net effect can be determined. Doing this, the following equation is produced, ignoring all small cross products,

$$m(H, t+1) >= m(H, t) * (f(H) / \bar{f}) * [1 - P_i * (\delta(H) / (l-1)) - o(H) * P_m].$$

Equation 2.12

Therefore, it can now be concluded that, highly fit, short-defining-length schemata (termed building blocks) are propagated through generations by being given exponentially increasing samples to the observed best: all this requires no special bookkeeping or memory other than populations of n strings. Further the computational processing is proportional to the population size, n, yet $n^3$ useful schemata are processed in parallel (implicit parallelism).

So by using the idea of schemata processing, the complexity of the problem can be effectively simplified, from being one of building high performance strings by trying every conceivable combination, to one where improved strings are incrementally constructed from the best partial solutions of past samplings.

Goldberg [12] makes the following remark with regard to problem coding and schema theory, highlighting the strengths and perhaps possible weakness of implementing a GA based optimising tool,

> The GA depends upon the recombination of building blocks to seek the best points. If the building blocks are misleading due to the coding or the function itself, then the problem may require long wait times to arrive at near optima solutions.

## 2.3 GENETIC PROGRAMMING

In this section, the way in which GP works is presented, differences between GP and GAs are highlighted, and the underlying theory of GP is introduced. Section 2.3.1 introduces GP and its functioning and section 2.3.2 presents the theory behind it.

### 2.3.1 Genetic programming

Genetic programming (GP) is an evolutionary computational technique that belongs to the evolutionary programming family. This family of methodologies evolves computer programs over time. Genetic programming derives from the GA, extending its abilities to allow it to produce computer programs directly. It was first proposed by Koza [21,22]. A drawback of using the classical GA is the need to have to re-state the problem in some kind of numeric form. This leads to a time-consuming process of trying to find a suitable mapping system, which also makes it difficult to relate the chromosome meaning to the problem. GP overcome this by allowing the chromosome to express the solutions as non-numeric forms such as computer languages. Traditionally the language used is LISP but almost any language can be used. However, the major difference between a GA and a GP is that the former uses fixed length chromosomes and the latter can handle cases that vary in both length and structure. Analogous reproduction, crossover and mutation operators (additional specific operators are also available) are used by the GP as well as fitness proportionality. However, the crossover and mutation operators must function such that they produce only syntactically valid programs.

The GP traditionally represents programs/chromosomes as list of instructions, expressed in pre-fixed notation (see Figure 2.6). This is attributable to the fact that it was originally designed with LISP (LISt Processing) in mind as an implementation language. The main reason LISP was selected was because it allowed for a list of instructions to be executed without any additional compilation or linking, this meant that the fitness of evolved programs could easily and quickly be determined. The alphabet (global function/terminal set) over which the GP is applied is problem specific

(i.e. the genetic material manipulated by the GP is largely dictated by the problem to be solved). This contrasts with the general applicability of the binary alphabet used by the GA, regardless of the problem. The GP alphabet, however, consists of only two classes of elements, functions (programming elements, which take one or more input parameters) and terminals (programming elements, which take no input parameters). The function class of elements in general allows for the sequencing and structure of programs to be established, while the terminal class of elements is required in general to act as leaves, marking the end of branches or indicating the initial action signifying the end of a structural path through a program. The effective defining of the alphabet and its cardinality, through the stating of functional and terminal requirements of a problem, gives the GP the chance to represent the problem in a form more closely linked to its natural language. This overcomes one of the most often touted drawbacks with GAs, that being the "art" of coding them and translating them back to the problem domain. However, the construction of the functions contained within the alphabet has an additional problem associated with them, closure. Closure is a property defined by Koza [20] that ensures that any combination of the functions and terminals defined for a problem, result in a syntactically correct program. This property states that all functions and terminals must return the same number of parameters (usually this is one).

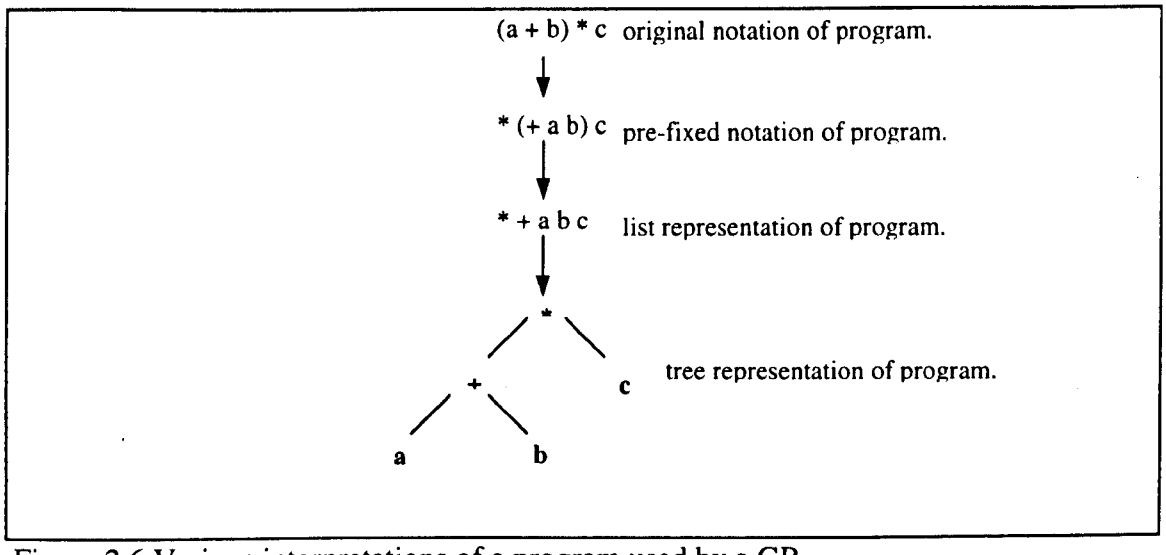



Figure 2.6 Various interpretations of a program used by a GP.

In order to foster code re-usage and aid in the reduction of the potential size of programs produced by GPs, Koza [21] introduced Automatically Defined Functions (ADFs). These are equivalent to procedures or subroutines within traditional programming languages. This allows the GP to contain two forms of code blocks, either ADFs or main bodies. In any program, there can be any number of ADFs but only one main body. Additional restrictions on the structure of programs are:

- All ADFs must come before (i.e. be defined before) the main body code block in the program list.
- An ADF can only call another ADF if the ADF to be called has already been defined (i.e. if it is to the left of the caller in the program list, assuming that the main body block is at the right end of list).
- All code blocks must have associated with them a single function/terminal set, this restricts their access to a subset of the functions and terminals (alphabet) defined for the problem.
- All the programs in every population of a run must contain the same number of ADFs, all in the same order.

Execution of a program always starts in the main body, which can then call any of the ADFs defined in the program. The formulation of a problem for the solving by a GP, therefore requires not only the identification of the alphabet of the problem, but also the identification of: how many ADFs are to be used; what the calling priorities should be; what subset of the global alphabet each code block should be limited to and what depth and width restrictions to place on each code block.

As mentioned earlier, the reproduction process for GPs is the same as that for the GA, so this will not be considered here (see section 2.2), however the structural differences of the chromosomes as well as the validity constraints placed on them give rise for need of functionally equivalent but structurally distinct crossover and mutation operators, which will be presented here.

There are two main forms of crossover, branch and context preserving. These handle the additional burden of ensuring syntax validity differently but utilise the same general form of information exchange to produce offspring. Information exchange during the crossover process takes the form of cutting and pasting different sub-trees between parents. The way these sub-trees are identified is the key differentiating feature between the two crossover methods. The application of each of these crossover methods offers either speed gains (branch-crossover) or greater degree of exploration, at the expense of speed, (context-preserving crossover). The application of these operators must ensure that the code block segments in the resultant offspring utilise only those instructions they are permitted to use and that all functions they call have the correct number of input parameters.

The common exchange of information these crossover operators utilise overcomes the syntax validity issue. This is achieved by exchanging whole sub-trees (i.e. when a point is selected for crossover, all the elements in its sub-tree including the node are copied then replaced by the copied sub-tree from the other parent). This effectively makes the crossover operation one of swapping single parameters (which may or may not have a sub-structure of functions and terminals associated with them). This avoids the possibility of elements within a sub-tree not being copied (leading to excess parameters in one offspring and insufficient parameters in the other) or single sub-trees being replaced by multiple sub-trees (leading to the overstating of the number of parameters used by the parent node).

In branch crossover the crossover sub-trees are identified by using a block restriction technique, that is information exchange can only take place between corresponding code blocks in both parents (e.g. information can only be exchanged between ADFn in both parents, where **n** must be the same for both parents or between the main bodies of both parents). This process ensures that the code blocks within the resulting offspring only ever contain those function/terminals they are permitted to use. The crossover process itself requires: that a point be randomly chosen within the range of the first parent, then the code block this point is in be determined, the second parent be restricted to choosing

a random crossover point within the range of its corresponding code block and then the information exchange process takes place. An example of this crossover process can be seen in Figures 2.7 and 2.8, here each parent contains a main body code block as well as a single ADF code block. The function/terminal set for the ADF (labelled ADF1) is {*, +, -, a, b, c, d, g} and for the main body code block {+, *, /, -, ADF1, a, b, e, k}. The program for the first parent is:

[ *ADF1*[ (a + b) * c - d] *MAIN*[ b + f * ADF1] ],

and for the second parent:

[ *ADF1*[ g * a * b] *MAIN*[ ADF1 / (k - ADF1) * a] ].

Assuming the crossover point chosen by the first parent, $C_1$, is 2 (corresponding to the '+' element) then this restricts the second parent to choosing a crossover point, $C_2$, within its corresponding ADF1 (i.e. its range for the crossover position is limited to 0 to 4). Assuming the second parent chooses 1 for its crossover point (corresponding to the 'g' element) then the crossover process will result in 'a + b' being exchanged with 'g'. The outcome of this can be seen in Figure 2.8.

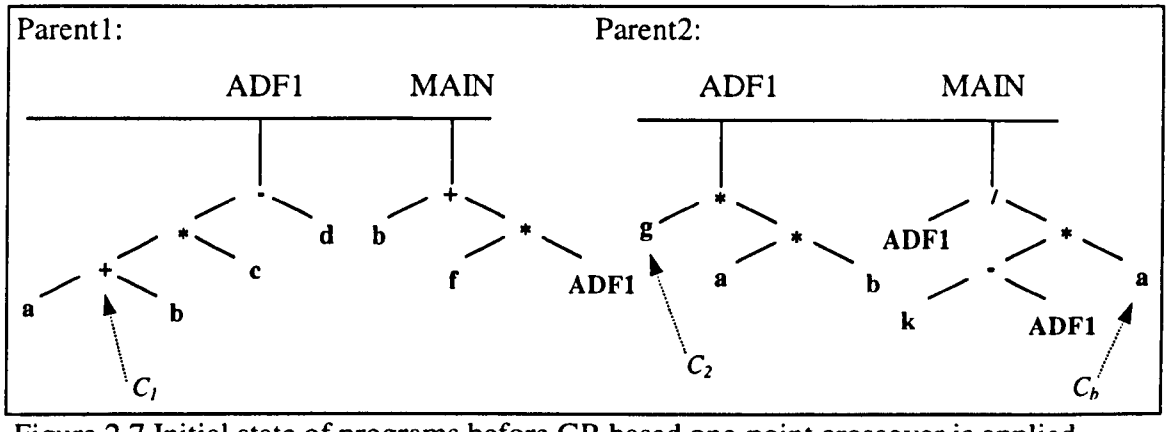



Figure 2.7 Initial state of programs before GP-based one-point crossover is applied.

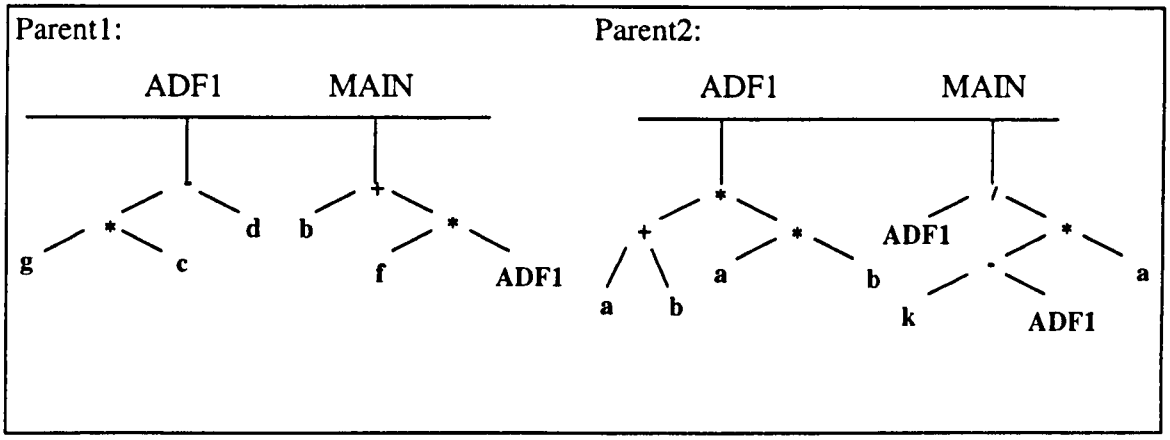



Figure 2.8 Resultant structure of programs after GP-based one-point branch crossover has been applied.

The identification of sub-trees for the crossover process in context-preserving crossover relies on functional restriction. Here any point within the whole program can be chosen so long as the exchange of sub-trees does not introduce any elements not permitted into a code block. Ensuring that the exchange of sub-trees does not create this situation is a time consuming process, however the benefit of this process, is more extensive exploratory powers for the GP. The process works by first selecting a random point, $C_l$, within the first parent's length. The sub-tree at this point is traversed and a list of all the distinct functions and terminals it contains is made. Using this list, points in the second parent's program are marked, signifying the fact that they are possible recipients for the selected sub-tree (i.e. the contents of the sub-tree is compatible with both theirs and the first parent's function/terminal set and parameter requirements). From these marked points one is chosen randomly as the crossover point, $C_b$. The two sub-trees are then exchanged giving rise to two new programs. Using the example in Figure 2.7, assuming the same crossover point in the first parent, $C_l$, then their exist 6 possible crossover points in the second parent, these are, all the points in ADF1 and the terminal element 'a' in the main code block. If the terminal element in the main block is chosen, $C_b$, then Figure 2.9 shows the resultant programs.

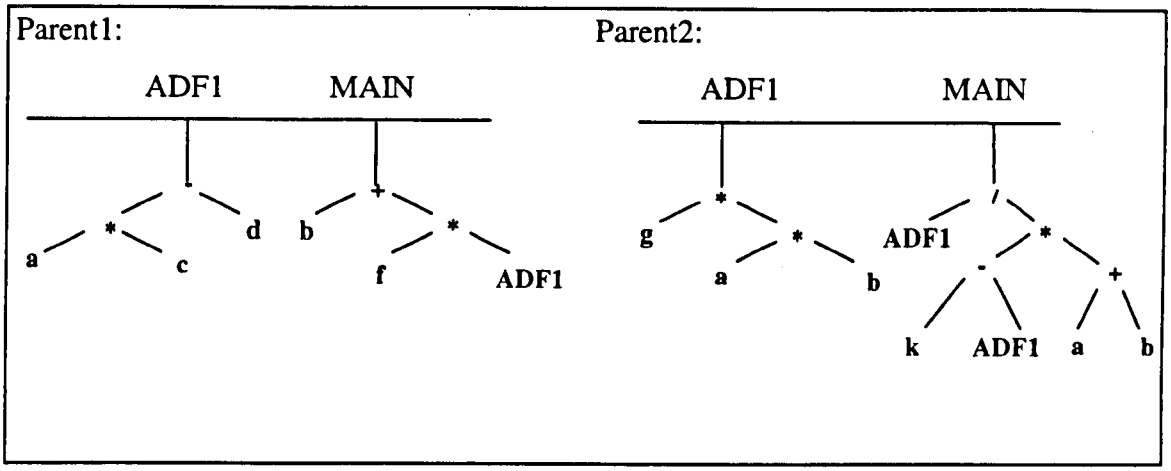



Figure 2.9 Resultant structure of programs after GP-based one-point context-preserving crossover has been applied.

The mutation, process is less problematic as regards syntax validity, so long as the same function/terminal set which was used to create the block is used for the pool in which the replacing genetic material is chosen from. The most common form of mutation is uniform mutation. Here mutation is equally likely to be applied to the leaves of a program tree as it is to the branches within it. The mutation process is applied to each of the code blocks contained within a program in turn. It requires that the list representing each code block be traversed one element at a time and a random decision be taken at each element as to whether to mutate that elements sub-tree below that element. If the decision is to mutate the sub-tree, then a randomly generated replacement sub-tree (of limited depth), produced using the code block's functional/terminal set replaces the sub-tree. The traversal processes then continues with the next list element after the newly created sub-tree entries. If however the decision is not to mutate then the next element in the program list is processed. The mutation rate governs the decision as to whether an element and its sub-tree should be mutated. An example of this mutation process can be seen in Figures 2.10 to 2.13, here the program:

[ *MAIN*[ + (b / c) * d - e] ],

is to have the mutation processed applied to it. Figure 2.10 shows both the list and tree representation of this program. Starting at the first element in the program list (and the each subsequent element in turn), a random number is generated and compared with the

mutation rate (MR) value to determine if mutation should take place. This gives rise to the following mutation process:

i) say at '+', the random generated value is greater than MR.

ii) move onto next program list element.

iii) say at 'a', the random value generated is less than MR.

iv) generate random sub-tree to replace terminal 'a' and insert the sub-tree into the sub-tree. Assuming the replacement sub-tree 'f * g' is generated then the program list will look as in Figure 2.11 (the replacement section in underlined).

v) move onto next non-newly inserted element in program list.

vi) say at '-', the random value generated is greater than MR.

vii) move onto next element in the list.

iix) say at '*', the random value generated is less than MR.

ix) generate random sub-tree to replace '*' and its associated sub-tree. Insert this replacement sub-tree into the program. Assuming the following replacement sub-tree is generated, 'h', then Figure 2.12 shows the resultant program.

x) move onto the next non-newly inserted element in the list.

xi) say at 'e', the random value generated is less then MR.

xii) generate random sub-tree to replace 'e'. Insert this replacement sub-tree into the program. Assuming the following replacement sub-tree is generated, 'i', then Figure 2.13 shows the resultant program.

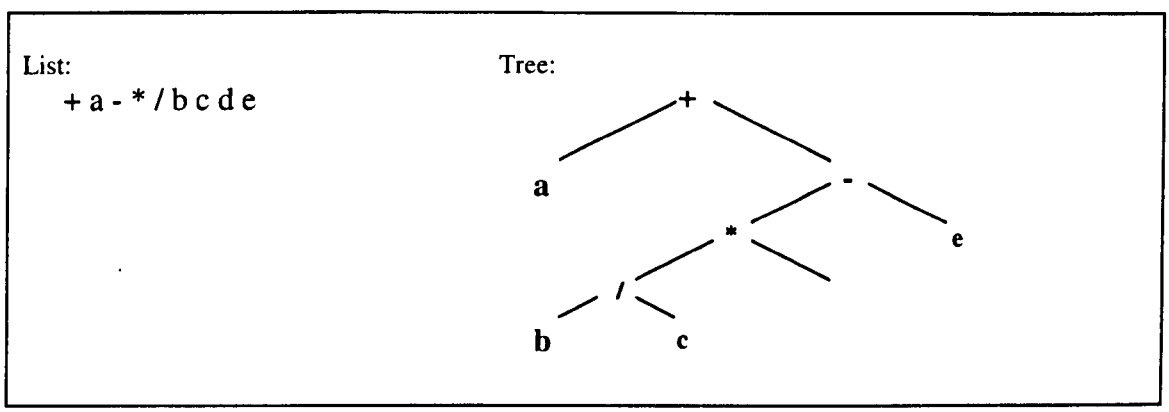



Figure 2.10 Initial program structure before GP mutation process commenced.

The resultant program produced by this particular mutation sequence is:

[ *MAIN*[(f * g) + (h - i)] ].

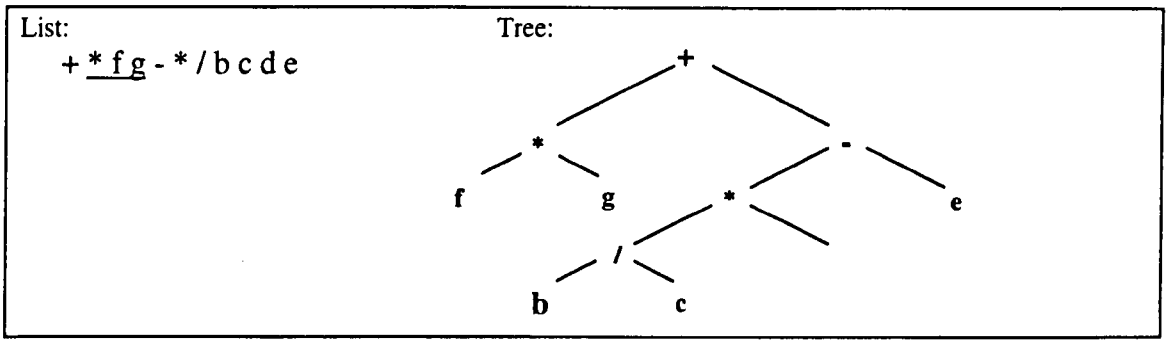


Figure 2.11 Mutation of a terminal into a sub-tree by GP mutate operator.

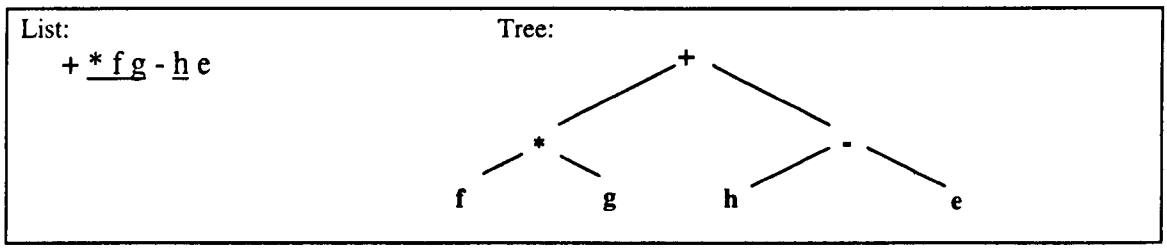


Figure 2.12 Mutation of a sub-tree into a terminal using GP mutation.

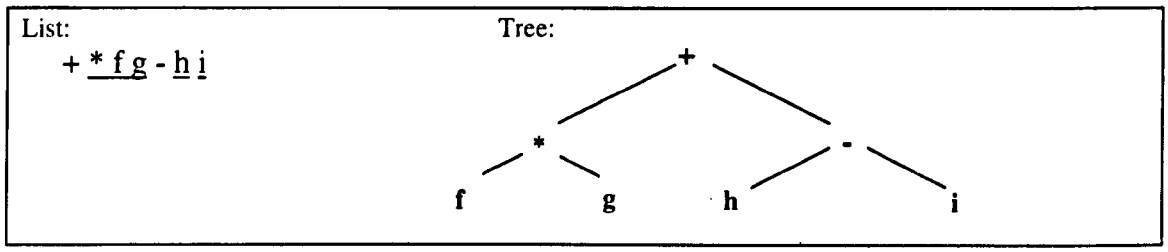


Figure 2.13 Mutation of a terminal into a terminal, using GP mutation.

Owing to its derivation from the GA as well as the recency of its inception much of the research on GP exploits work done in GAs, however there are investigations into new areas pertinent to GP, such as: multi-typed programs (Montana [23]), stack manipulation (Keith and Martin [24]), abstraction, memory (Teller [25]) and structured programming (Pringle [26]).

Two methods have been used to determine/produce generality in programs, program length and noise. Koza [21] and Kinnear [27] augment program length to the set of

existing constraints defined for the problem to produce a composite fitness. This approach is used since in some cases the time taken to generate a result is as, or possible more, important than its correctness. So, by minimising the length of programs, the chances of over fitting the evolved programs to the test set are reduced, resulting in more generally applicable programs. However Koza [21] found that there was a substantial increase in the number of evaluations required to evolve near optimal programs when program length was used as a fitness factor. Kinnear [27] found that more general solutions were produced if a factor inversely proportional to the program length was used in the fitness. Reynolds [28,29] introduces a noise component into the evaluation process used to evolve programs, first via applying random noise into the environment to produce controllers for steering a robot through corridors also by using multiple test cases in the development of a visual based guidance controller. By using either of these methods (random noise and multiple test cases) as the basis of determining the fitness of a program, over-specific or complicated programs as well as opportunistic ones are discouraged, in preference for more globally applicable solutions leading to the evolution of more robust programs.

A number of alternative crossover and mutation operators have been proposed. Two however of relevance to the work presented in this thesis are modular-crossover and constant perturbation. Kinnear [30] proposes the modular-crossover operator. This operator allows for sections of code in the middle of sub-trees to be exchanged. It was found that the use of this operator resulted in performance gains over the basic GP (i.e. GP with no ADFs). However, the operator requires additional checks to ensure that the resulting offspring do not over or under state the number of formal or actual parameters for a function. Spencer [31] introduces a mutation operator (constant perturbation operator) that only affects special terminals, termed ephemeral random constants (fixed constant whose initial value is generated randomly). This operator multiplies all ephemeral random constants in a program in turn by a random value in the range of 0.9 to 1.1 (i.e. it perturbs the constant by up to +/- 10% of its original value). It offers two advantages, firstly, it allows for "fine tuning" of constants or coefficients in programs (the coarseness of the crossover and mutation operators does not allow for any

incremental development of constants, since they either copy the constants unchanged elsewhere or simply overwrite them). Secondly, it increases the diversity of constant values in the population, leading to a more effective traversal of the constant search space of a problem.

### 2.3.2 Genetic programming theoretical framework

The development of GP theory is still in its infancy, as an area of research. In his initial book, Koza [21] tackles the theory issues indirectly by drawing parallels with the GAs schema theory, although no proofs of these parallels are given, the reformulation concepts he outlines in this book are currently the most widely quoted theory for GP.

The theoretical framework for GAs is outlined in section 2.2.2 and is based on the manipulation of similarity templates, schemata, within strings (chromosomes) as well as their associated growth and decay rates throughout the optimisation process. For this theory to be fashioned for GPs it must take into account the differences in structural representation of chromosomes that exist. GAs have a linear chromosome structure as such their schema poses a linear structure. GPs have a hierarchical (tree like) structure and as such their schemata representation should reflect this and as a result modification be made to the general schema theory where necessary, to encompass these representational and associated functional differences.

Koza [21] defines the similarity template (schema) for GP, as the set of all individual trees from the population that contain, as sub-trees, one or more specified sub-trees. This gives rise to an infinite number of such trees. However, GP always place restrictions on the initial or resultant length of processed programs, this effectively reduces the set of trees that contain the specified sub-tree to a finite amount. The average fitness of a schema is in effect the average fitness values of all the individual trees belonging to its schema.

The growth and decay characteristics, as a result of fitness-proportionate reproduction, of schemata are independent of the character of the individual objects in the population. As such the GA's reproduction growth equation applies here. That is the equation

$$m(H, t) = m(H, 0) * (1 + c)^t, \qquad \text{Equation 2.13}$$

defines the growth and decay of schema regardless of their structure. This means that schemata in GP also experience exponential growth and decay rates proportional to their fitness level in regard to the population average. So above average schemata experience exponentially increasing samples in future generations and below average schemata experience exponentially decreasing samples.

Koza identifies two crossover schema processing cases, first where the schema is defined in terms of a single tree and secondly where it is defined as containing more than one specified sub-tree. In the first case Koza states that crossover is less disruptive on these schema when they are compact in size and as a result sub-trees (i.e. sub-programs) from highly fit programs are used also building blocks for constructing new individuals. This use of highly fit sub-trees as building blocks holds true for the second case, however here it is defined in terms of relative number of points and number of encompassing trees. For those schemata whose sub-trees have a relatively small number of points defining them and which have a minimal tree encompassing all their disjoint sub-trees which is relatively small, will be used as building blocks for constructing new individuals.

The combined effect of reproduction and crossover is again the same as in the case for GAs, that is,

$$m(H, t+1) >= m(H, t) * (f(H) / \bar{f}) * [1 - P_i * (\delta(H) / (l-1))]. \qquad \text{Equation 2.14}$$

As mentioned earlier the area of GP theory is still very much in the initial stage and as such this schema theory approach is somewhat provisional and possible incomplete to some degree. In this regard, Langdon and Qureshi [11] highlight papers critical of this paralleling with GA schema theory, showing that the building block hypothesis does not

hold true if an omitted case of crossover schema processing is included. However, none of these papers disputes the power and potential of the GP, just the mechanisms that allow it to do so.

## 2.4 SUMMARY

This chapter has presented both the functional logic and theoretical framework of GAs and GP and has highlighted literature in each field relevant to their implementation in this work. As mentioned in the introduction, it is not the aim of this thesis to comment on or add to any these areas. Rather, it seeks to determine their applicability in the production of effective and robust robot control systems. The generality of these methods and the efficiency with which they search makes them potentially ideal tools to accomplish this aim.

# Chapter 3

# LITERATURE REVIEW

## 3.1 INTRODUCTION

The concept of autonomous mobile robots has been at the heart of much science fiction literature for many years. As a result it is easy to envisage that these devices are actually intelligent and display human like competencies. However, if we consider some of our everyday actions, which we perform instinctively, such as moving around without colliding with objects, recognising objects and communicating with each other, then these are actually complex tasks and are proving extremely difficult to replicate in mobile robots.

It has already been stated that this thesis is to investigate the ability of Gas to plan paths for mobile robots and GPs ability to evolve communication strategies for teams of robots. This chapter thus presents a review of the literature in these areas. In particular section 3.2 covers path planning and section 3.3 reviews work concerning the use of communication in robot controllers.

## 3.2 PATH PLANNING

Path planning is the process by which a route is produced for a robot to traverse, taking it from a source point to a destination point. It is required that this process produces efficient paths in terms of time, distance, energy or some other appropriate metric as well as ensuring the robot's safety. The approaches proposed in the literature to path planning can be categorised as either potential field or free space approaches. These are reviewed in sections 3.2.1 and 3.2.2 and a review of evolutionary approaches to path

planning is covered in section 3.2.3. Finally a summary of the shortcomings of these approaches are given in section 3.2.4.

### 3.2.1 Potential Fields

Khatib [32] introduced artificial potential fields as a method of robot path planning. These fields work on the premise that a force can be used to direct the movement of an object. An attractive field can be used to direct it to a desired location whilst a repulsive field can direct it away from an undesired one. So if the destination of a robot is considered as the point of attraction, and the obstacles within the environment as emitting repulsive forces a method for planning collision free paths by utilising the resultant forces can be implemented. The attractive field extends throughout the environment, whilst, the repulsive field is limited to an area around the obstacle. This effectively defines the clearance region around the obstacle. Defining the environment to be the superposition of these fields' forms the resultant artificial potential field.

Two categories of potential fields exist. Firstly those which use the field to define the points in the environment through which the path passes (Kim and Khosla [33] and Barraquand and Latombe [34]) and secondly those which use the field produced as control inputs to drive the robot (Khatib [32] and Ikegami and Ozono [35].) In the latter, path planning and collision avoidance becomes a low-level process instead of the customary high-level process, which is several magnitudes slower. The resultant increase in performance offered by the low-level process allows for real-time path planning and obstacle avoidance.

Whilst an attractive idea, potential fields exhibit several drawbacks as documented in Koren and Borenstein [36], the main ones being:

- The existence of multiple local minima within the field. These are caused by cancellation of opposing forces during field combination, which results in null driving forces or potential wells.
- The inability of guaranteeing convergence to the goal in the presence of local minima.
- The difficulty of generating globally optimal paths.

The improvement in robustness of the potential field approach, which has been the main focus of recent research, requires the overcoming of either of the first two drawbacks listed previously. Khosla and Volpe [37] tackled an additional problem to that of Khatib [32], the modelling of obstacles through their repulsive forces. Khatib uses an inverse quadratic function, which results in potentially infinite forces, which require infinite control effort. Khosla and Volpe [37] propose the use of superquadratics -a more computationally expensive function- to overcome the problem. Their approach did not totally eliminate local minima but was effective within uncluttered environments.

Adams, Hu and Probert [38] suggest the temporal relocation of the robot goal when local minima are encountered. Here they show that by changing the position of the goal slightly when a local minima is encountered allows the robot to escape them. Arkin [39] on the other hand introduces a persistent low level of random background velocity directions, in the form of noise, which only becomes significant at equilibrium points, which results in the robot performing random movements. A similar idea is adopted by Barraquand and Latombe [34], in which Brownian motions are executed to escape any minima. These methods are only probabilistically determinant, so do not guarantee that the robot will not be trapped by a local minima. Warren, Danos and Mooring [40] again reduced but did not totally eliminate local minima by using a two-stage process. First a trial path is chosen, then each point along the path is influenced simultaneously by a potential field. Using this method leads to more effective global planning. Ikegami and Ozono [35] did not attempt to develop a potential function devoid of local minimum but instead tried to tackle the second drawback. Their function used the distance value

model and utilised four forces instead of the standard two. These additional forces were rotational and virtual attractive forces. The aim of the rotational force was to escape local minima whilst the aim of the virtual attractive force was to smooth out paths. By using a combination of basic navigation and wall following navigation local minima which occurred on or near the boundary of arbitrarily shaped obstacles could be escaped from. Masoud and Bayoumi [41] used vector potentials to address the same problem. By combining them with scalar potential fields, motion can be directed along the family of equipotential surfaces, which are normal to the flow, there by increasing the controllability. Their approach was able to escape from local minima and handle cluttered environments. Barraquand and Latombe in [34] adopted a well filling approach when local minima were encountered. This, too, is a deterministic method for escaping local minima.

The construction of potential fields without local minima was first reported by Rimon and Koditschek [42] and Kim and Khosla [33]. The Rimon and Koditschek method transformed the environment to a spherical representation and then exploited its geometric properties to develop a general navigation function. The Kim and Khosla method however utilised the principles of harmonic potentials to produce the potential field. However, the potential functions of these approaches can only handle simple geometric shapes and the computational effort to generate the fields can be high. Schmidt and Azarm in [43] used an unsteady diffusion equation to generate their potential field. It works by assuming that some gaseous substance diffuses throughout the environment (instant absorption of the gas by the obstacles is assumed.) The concentration at the goal is constant and lower than all other areas. Time must be allowed for the concentration distribution of the gas to reach its equilibrium state. At this point a monotonically decreasing concentration from the current position to the target position is established through the substance filled space indicating the path (non-optimal paths are produced if planned before equilibrium position is reached). High speed path planning can be achieved using this method. Akishita, Kawamura and Hayashi [44] proposed the use of hydrodynamic potentials. This is based on the concepts of fluid flow. The environment is assumed to be closed and to contain a fluid.

The goal is treated as the suction point (hole) within the environment, such that any particles floating on the surface of the fluid will flow towards the goal. This allows the environment to be represented as a flow field, which is described as a kind of velocity potential.

All of the aforementioned methods utilise some potential like function, which either possesses some properties of, or can be rewritten as a Laplace equation. However Barraquand and Latombe [34] introduce a method which does not rely on these equations and one which can handle complex geometric shapes and whose computational performance is very good. Their method uses wavefront expansions emitted from obstacles and the environment's boundary walls to generate a potential field for individual control points. The result is a potential field with only one minimum and a path being found if one exists. However, the paths produced tend to keep too far away from obstacles. A hierarchical bitmap resolution of the environment is used for the purpose of path finding, which enables the minimum amount of information to be considered for generating paths.

The extension of the potential field technique into dynamic environments takes several forms. Ratering and Gini [45] use a hybrid potential field, which consist of a static field, based on Barraquand and Latombe [34], to represent the known floorplan, and a dynamic field for the local information sensed by the robots sensors. These two fields are combined to produce the potential field on which navigation is based. A histogram grid is used to distinguish between stable and moving obstacles. This allows for unknown stable objects to be added to the floorplan and previously stable objects to be removed and the floor plan potential re-computed. Local minima, which are encountered, are overcome by either waiting for the obstacles to move or reducing the extent of the obstacles fields. The algorithm however, does not eliminate all collisions with moving obstacles.

The diffusion strategy used by Schmidt and Azarm [43] caters for avoiding unknown moving obstacles, moving goal points and obstacle tracking. Moving objects can,

however, cause transient local peaks in the distribution field as they pass by the robot, causing a temporary planning uncertainty as the steepest ascent algorithm tracks these peaks instead of the goal. To resolve this uncertainty the robot must wait a short period of time for the distribution to disappear. Akishita, Kawamura and Hayashi [44] and Akishita, Hisanobu and Kawamura [46] showed that a navigation function based on hydrodynamics could plan paths for a robot in the midst of one or two moving objects operating in a bounded free space. This however proved vulnerable to the wakes produced by moving objects, which resulted in temporary local minima.

Borenstein and Koren [47] apply a potential field directly to the sensor readings generated by the robot, giving a reactive or reflexive navigation method. These sensor readings, in the form of incremental certainty values are represented in a 2-d Histogram grid. Each of the cells in the active window around the robot exerts a virtual repulsive force towards it. This force is proportional to its active window position and inversely proportional to its distance from the robots centre. These virtual forces are summed together to produce a resultant repulsive force. This resultant force is summed with a virtual attractive force of constant magnitude, which is pulling the robot to its goal to produce the net resultant force vector. This virtual force field approach worked well in sparse environments.

Manz, Liscano and Green [48], showed that the use of the generalised potential field performed better than the classical potential field when applied to reactive navigation. The generalised potential proved more stable when avoiding obstacles and could pass through doorways and along narrow passage ways (although its performance did deteriorate with speed.) Both these methods had problems avoiding directly approached obstacles. However the biggest problem noted was the setting of the appropriate gain factors for each algorithm.

Summarising, a lot of research effort has been directed towards potential field based path planning. However, whilst computationally efficient, all of the aforementioned reactive potential field methods suffer from some or all of the following problems:

- oscillations in the presence of obstacles as well as in narrow passages.
- difficulty passing through closely spaced obstacles.
- susceptibility to local minima traps.

### 3.2.2 Free Space Methods

Path planning using the potential field approach requires that the environment be modelled in terms of attractive and repulsive forces. An alternative approach, is to model the environment in terms of free space (sections of the environment which contain no obstacles), which can either be achieved by modelling the free space as a continuous expanse or as a number of tessellated shapes. The subsequent processing of the free space both to identify regions the path will pass through as well as to generate the path itself illustrates the dual step nature of this class of path planners. Class members are internally differentiated from each other using the following criteria:

- the way free space is modelled,
- the way the free space model is processed to find the path.

The majority of the free space modelling techniques tend to fall into what is called the graph search class of path planners, owing to the way the free space model is processed, which is by using some heuristic (or otherwise) based search technique.

The most common form of modelling free space is the configuration space (C-space) approach as presented in Lozano-Perez and Wesley [49] and Lozano-Perez [50]. It shrinks the robot to a point while at the same time expanding the obstacles relative to the robot's shape. The resultant free and accessible points are then mapped into a form which is a continuous expanse. This C-space approach is best suited to situations where robots have translational motion only, for inclusion of rotational motion adds an extra dimension to the expanded obstacles for each degree of rotational freedom offered. This introduces the need for more complex and computationally expensive geometric

algorithms. Lozano-Perez [51] approached this problem by approximating the C-space obstacles using sets of slices whose dimensions were one less than the number of degrees of freedom of the robot. This still allows collision free paths to be found by searching a vertex graph.

C-space modelling suffers from two drawbacks. First and most significantly the computational complexity of the mapping between the real world and the model and, secondly, increased path search times caused by the large dimensionality of the model.

The complexity of the model mapping process was addressed by Warren, Danos and Mooring [40], in which an environment consisting of polygonal obstacles is mapped into the configuration space by tracing in turn the boundaries of the obstacles. The approach of Hara and Nagatas [52] allows for spherical approximations to obstacles, which enables computational requirements to be reduced. Tso and Liu [53] generate the configuration space by solely processing the contours of the obstacles.

The standard search process for the C-space model is accomplished by using a visibility graph. However as mentioned previously, this method results in long search times. As a consequence an active area of research is how to reduce search times. Most of these search methods are based around the cell decomposition approach. Using the decomposed cells a connectivity graph is constructed which represents the adjacency relation between the cells (Schwartz and Sharir [54] and Brooks and Lozano-Perez [55].) Searching this graph then generates the path. Zhu and Latombe [56] use an approximate cell decomposition method, which decomposes the configuration space into rectangloid cells, which are labelled as empty, full, or mixed. Mixed cells are recursively decomposed if encountered and augmented to the graph. This leads to significantly faster path planning. Quinlan and Khatib [57] also use approximate cell decomposition. However, their method is based on a new class of cells called Slippery cells, in which the cells conform to a general shape and are constructed such that the outward normal from any point on its boundary does not intersect the cell itself. Hence, this approach maintains the property that a path using a local algorithm can connect any

two points, which belong to the cell. The use of these cells enable the free space typically to be more efficiently represented than in those methods which use rectangular cells, resulting in smaller connectivity graphs and faster path planning. These approximate methods have an inherent drawback, which is that the approximations they make may result in the planner failing to find a path when one does exist.

This concept of cell decomposition, using simple geometric shapes to represent free space, has been applied by other free space modelling approaches. Brooks [58] uses overlapping generalised cones, formed by sweeping a 2-dimensional cross-section along a curve in space and deforming it according to a sweeping rule to decompose the free space. This approach limits moving objects to convex polygons and obstacles to the union of similar polygons. The algorithm, despite outperforming the configuration space, cannot handle cluttered environments nor can it easily represent large free spaces. Liu et al. [59], Noborio, Naniwa and Arimoto[60] and Wong and Fu [61] use a quadtree to model the environment, by decomposing it into four equally sized squares by splitting the area in half horizontally and vertically. Any resulting decomposed areas, which contain both free points, and obstacles are decomposed further. The path is generated in a two-stage process using the quad-tree. First by selecting a sequence of several intermediary positions and then generating collision-free motion by trying these intermediary positions sequentially. An extension of the quad-tree model is that of the octree. Here the space is recursively decomposed into 8 cubic areas instead of the 4 square areas. The collision-free path is found here by heuristically searching the free space. This model is used by Fujimura and Samet [62] and allowed 3-d environments to be represented and searched. Tokuta [63] decomposes and models free space within an environment which contains polygonal obstacles, using binary space partitioning. This requires the environment to be first tessellated into cells with convex shapes which forms representations for free and obstacle spaces. A binary space partitioning tree is then formed from this by recursively processing each of the edges that define polygonal objects and identifying which areas of the space they partition are inside or outside of some object. By using the union of outside cells, an adjacency free space connectivity graph can be formed. The path is generated by searching for the free space regions that

will result in the best path and ensuring that it passes through the clearance point of any obstacle line it meets. Habib and Asama [64] introduce a technique based on the free link concept which models the free space between obstacles in terms of convex areas, which are formed by joining the vertices of the polygonal obstacles (effectively giving a free link) subject to a set of rules. The midpoints of all free links formed are calculated and used as nodes in what is termed a Maklink graph, which is then heuristically searched to determine the collision-free paths. This approach reduces the complexity and size of the graph to be searched producing a more efficient planning algorithm.

An alternative to the decomposition technique is the modelling of the environment as an expanse of free space accessible from a given fixed point. Applying to this free space some kind of traversal cost function allows for the indication of various kinds and shapes of terrain as well as a way for generating collision-free paths in a quick and efficient manner. Wave propagation is one such technique. Hughes,Tokuta and Ranganathan [65] introduce a new propagation algorithm which uses weighted regions. The free-space in the environment is divided into regions with homogenous traversal cost, which are then processed. First the regions are covered by an initial set of propagations, repeated propagations are then performed until there is a convergence to a solution for each region. An admissibility criterion is applied to the paths as they are propagated ensuring that paths, which cross higher weight regions, will not be chosen. These wave propagations emanate from the destination point and then spread throughout the free-space, propagating the Cartesian distance between neighbouring regions and the cost of traversing a region. A path is formed by following the local path segment to its sub-goal, then traversing the paths to subsequent sub-goals until no more remain, indicating the goal has been reached. Zelinsky [66] introduced a propagation algorithm which addressed the endemic problem of propagation methods, obstacle clearance. Typically, propagation methods produce paths, which pass too closely to obstacles. This improvement is accomplished by using two types of propagation fields. First a distance propagation is applied to free space where the discrete city block distance of the points from the goal are propagated. Second an obstacle propagation, here the distance of the free space points from the points in the obstacles are propagated. By varying the extent

of propagation the clearance distance of obstacles can be changed. Combining these two propagations gives a path propagation, which is used to generate the paths. Following the steepest descent gradient from the start point to the goal forms paths.

With the exception of wave propagation, all these alternative modelling methods along with the configuration space refinements do however still require a large amount of computational effort in order first to model the environment appropriately then generate the corresponding graph representation.

The extension of free space based modelling algorithms to dynamic environments can be accomplished in several ways.

Liu et al [59] propose a new algorithm which uses a prioritised two-level planner. Collision free co-ordination is achieved by constructing a Petri-net and searching sequences of firing transitions. Fujimura and Samet [62] present an octree based navigation system for a 2-d environment. Both moving and non-moving polygonal obstacles are projected into a 3-d world, where the additional dimension is time. A collision-free path is planned by avoiding these projections. Fujimura [67] introduces the concept of accessibility graphs, which defines a graph on a set of moving objects, to produce time-minimal paths for polygonal obstacles moving with constant speed. Further, Fujimura [52] uses space-time diagrams and path search in a time-varying graph to generate time minimal times for a robot moving along a network of paths in the midst of polygonal obstacles. Shih, Lee and Gruner [68] represent moving obstacles in space-time as polyhedra and generate a graph based on the polytope representation of free space. A global search of the resulting graph produces the path to which a family of trajectories is generated using linear programming. Here obstacles move with constant speed but can move faster than the robot. Pan and Luo [69] present a technique based on traversability vectors, in which a path is divided into a number of segments according to some time resolution factor. Using a maximum swept area technique possible collisions can be detected which results in a time constraint for an area. A collision-free path is planned by regenerating a path such that its segments avoid areas with time constraints.

Noborio [70] evaluates several planning strategies for uncertain environments, which work by heading in a straight line towards the goal and traversing the boundary of any obstacles encountered until a leaving point is reached, after which motion continues in a straight line towards the goal. The determination of the leaving point of an obstacle boundary differentiates the algorithms and determines the kind of environments they can handle. Noborio and Hashime [71] extend this idea to cope with dynamic workspaces, in which it is assumed that the target robot can see the outline of all other robots. By defining an algorithm based on asymptotical decrease of the Euclidean distance between the target robot and the goal, the target robot can leave the boundary of any robot obscuring its path to the goal as early as possible despite any selfish movements on their behalf.

All the aforementioned methods require that knowledge of the paths of dynamic objects be known prior to motion planning. Borenstein and Koren [72] present a reactive system based on a vector field histogram, in which sensory data represented in a 2-d histogram grid is transformed into a polar histogram, representing the sensed obstacle density of the robot's surroundings. Using this, potential target areas whose object density is below a certain threshold are identified. From these areas the one whose heading most closely matches the direction of the goal is selected. This method suffers from oscillations in passageways, which may also occur for a while when obstacles are approached. Bennamoun et al [73] introduce an approach based on proximity sensors, in which the robot heads in a straight line towards the goal, using two sensors (a short-range one to detect obstacles and a long-range one to avoid obstacles). The robot is able to avoid collisions and reach the goal by determining the angles of the left and right boundary walls of obstacles and deriving the appropriate straight-line path. This method, however, has difficulties in determining openings in long walls and also in very cluttered areas. Noborio [74] presents several algorithms based on a method in which the target robot heads in a straight line to the goal whenever permissible. If other robots are sensed within its visible area, then the boundary of the nearest one is traced and an appropriate leave point searched for, in which the Euclidean distance to the goal decreases asymptotically. Wang et al [75] adopt a similar approach to Noborio [70,74] but also

introduce the idea of dynamic space traversal costs. Here points in the environment have traversal costs associated with them, which however change in the presence of moving obstacles. The cost of a point also varies in relation to how the robot passes it. With those points which will result in dangerous manoeuvres to get to or put the robot in a dangerous position being giving the worst traversal cost. Collisions with obstacles can either be avoided by waiting (in cases of moving obstacles) or by tracing the boundary of an obstacle to find a leaving point (in the case of static obstacles.)

Summarising, the major shortcomings of free space methods have been identified as:

- High computational demand for mapping between real world and model.
- Long search times due to large dimensionality of model.
- Approximation errors leading to the failure to find paths when they exist.

### 3.2.3 Evolutionary path planning

An emerging area of research is the application of evolutionary methods to path planning. Davidor [76] applied genetic algorithms (GA) to path planning, proposing the use of an analogous crossover operator to aid path generation. The exploration capacity of this method is dependent on the areas visited by the population of paths, so if a region of the environment has no path that passes through it then there is a very low chance of its being explored using this method. Shibata and Fukuda [77] apply a GA-based planner to generate paths in an environment modelled as a Maklink graph. The path is represented as a series of graph node points. Collision avoidance of multiple robots is handled by using a fuzzy logic based approach to indicate the areas which are most susceptible to collisions and re-planning paths appropriately. Griffiths et al [78] present a GA-based path planner which is embedded in a micro-controller. Here the free space is traversed by a series of interlinked paths which define areas of possible motion. The GA is applied to searching the resultant graph formed by these interlinked paths to produce a collision free path. Pigeon holing was employed to speed up the planning process. Michalewicz [79] introduces a GA-based planner for dynamic environments,

which represents its path as a series of via points. Using this representation an initial path is planned off-line and an on-line planner is used to produce local detours when unknown obstacles are encountered. This method requires the environment to consist of polygonal obstacles. Further, it does not make the most efficient use of its genetic material. The use of high penalty values for colliding paths prohibits them from contributing information to the next generation allowing possible useful information to be lost. Mazer et al [80] present a GA-based planner which uses a parallel machine consisting of 128 transputers. The planner has two stages to its application. First, landmarks are sequentially placed in the environment, then checks are made from each landmark to see if the goal is visible from them. The planning process ends when a landmark is placed within visual range of the goal. A path between this landmark and the goal is constructed. A parallel GA is used for both the landmark checking and to search a tree of the remaining landmarks constructed to produce the remainder of the path. A standard GA is used to maximise the distance between placed landmarks.

### 3.2.4 Path planning review summary

A review of the research literature concerning mobile robot path planning has been presented in sections 3.2.1 to 3.2.2. Although the techniques tackle a broad spectrum of the issues involved in robot co-ordination some general shortcomings of techniques exist. These can be stated as follows.

- The computational demands of the free-space techniques preclude their use in anything other than well-known static environments and relegate it to an off-line technique.
- Potential fields, whilst being computationally efficient, suffer from local entrapment as well as deriving less globally optimal paths.

Further, for dynamic environments the following shortcomings are highlighted.

- Approaches all assume constant velocity for robots.
- The shapes and paths of moving obstacles need to be known beforehand.
- Path oscillations in corridors or on the approach to obstacles are possible.

These conclusions, together with the results obtained by other researchers using GAs form the basis for a program of work aimed at investigating the applicability and potential of evolutionary computation to overcome the aforementioned problems.

## 3.3 ROBOT COMMUNICATION AND CO-ORDINATION

The role of communication within multiple robot systems has received little attention in the literature. More emphasis is given to basic competence acquisition suggesting that communication plays an incidental role. Berger et al [81] however argue that the traditional definition of behaviour-orientated control architectures as proposed by Brooks [82], Steels [83] and Verschure et al [84] is inappropriately defined and highly restrictive. Berger claimed that what these authors term as behaviour-orientated is actually reflex-orientated and it is this that fundamentally limits the architecture's scope for ingenuity, flexibility and applicability. Berger et al [81] proposes an alternative architecture which is divided into three parts: a behaviour system; a memory and judgement system, and an inter-robot communication system. This incorporation of communications indicates that communication between robots is not just a peripheral aspect but one, which has an essential and integral part within a multiple robot control system. The remainder of this chapter is dedicated to reviewing the current literature in this area. Section 3.3.1 considers the use of communication within traditional robotics, section 3.3.2 concentrates on evolutionary approaches to communication and section 3.3.3 summarises this section of the review.

### 3.3.1 Communication and co-ordination.

The majority of the literature in this area is concerned with collective robotics, where groups of either homogeneous or heterogeneous robots work co-operatively to achieve a global task. Such systems include SWARM (Jin et al [85]), CEBOT (Fukuda et al [86]), ACTRESS (Asama et al [87]) and GOFER (Caloud et al [88]). Each system has its own set of characteristics and communication requirements. SWARM systems require the communication of state information between nearest neighbours in their distributed environment, CEBOT systems utilise a hierarchical architecture in which only the master cells communicate in order to co-ordinate subtasks. ACTRESS uses multiple levels of abstraction for communication protocols, allowing for group casting and negotiation and the GOFER system uses a centralised task planner to communicate with the robots. Co-operation is the key nature of these systems. McFarland [89] defines co-operative behaviour as the result of interactions between selfish agents in order to maximise individual utility. This can be compared with eusocial behaviour, which is motivated by an innate need to survive, not a desire to co-operate. The term co-operative task used in collective systems is much looser than this and is concerned more with the performance gains that are achievable from robots working together and not what motivates them to co-operate.

Cao et al [90] states that a communication or control structure is a key pre-requisite to achieving co-operative robots. Dudek et al [91] argue that when there exists a need in a task for synchronised behaviour/actions between group members, some degree of communication is required between them to achieve this synchronisation. Both these assertions highlight the importance of communication within multiple robot systems. However, different forms of communication are possible, some of which are extremely subtle, while others are very formal and well structured. Cao [90] identified three general classes of communication:

- Implicit.
- Interaction via sensing.

- Explicit.

Implicit is the simplest form of communication in which the environment itself is used as a form of shared memory, with interactions between robots updating it and no deliberate attempt being made to communicate with others. An example of this is stigmergy, in which uncoordinated actions of robots in an environment lead to the solution of a global task. Another more subtle example can occur in tasks which are designed with no deliberate or intentional communication ability but, owing to the fact that individuals in the environment share a common object, a common local communication channel exists. From this, properties of the other robots such as: force, torque, orientation or distance can be derived. Work falling into this class has been produced by Arkin [92], Steels [93], Beckers [94] and Sen [95].

Communication based on interaction via sensing requires the robots to have some ability to discriminate between each other as well as other objects in their environment. Here local interactions between agents as a result of them sensing one another give rise to collective behaviours. Toquenaga [96] applies it to flocking and foraging behaviour of Egrets, where the concentration and distribution of other egrets is a key factor in determining food location. Hodgins et al [97] and Brogan et al [98] use relative distance between individuals in a local visual region to create dynamic herding behaviours amongst a population of robots. Kube et al [99] state that certain sequential tasks can be performed without the need for explicit communication or co-operation. The sequential tasks are modelled via a hierarchy of finite state machines. So the problem now becomes one of controlling state transitions in a set of asynchronous processes in a decentralised manner. This is achieved by using perceptual cues as the trigger for transitions between different tasks or to activate a specific behaviour within a sub-task controller. Mataric [100] proposes a set of basic interactions which can take place between robots such that the process of designing group behaviours for mobile robots is greatly simplified and more structured.

Traditional multi-agent tasks typically only require co-ordinating communication either at the start or end of the task. Here each member operates independently of the others until the end or from just after the start of the task. These tasks are typically highly parallel and tend to be those which can be accomplished without any explicit communication. Such tasks have reduced complexity and increased reliability as their key benefits. However, they do not maximise the performance of individual members of the group which results in possible sub-optimal performance. Further, where there are subsets of the group which do not communicate, performance can be probabilistic which in general is unacceptable.

Explicit communication is where a deliberate attempt is made to convey information to others. This can either be done by a central co-ordinator or distributed individual robots. The communication mechanism used here is critical to the practicality, efficiency and reliability of the group. Many of the issues of importance here are those affecting the networking field i.e. topology design and protocols. Here in particular, the use of overly rigid topologies or controls is of significant importance since they offer the greatest potential for brittleness in the system [91]. Ueyama et al [101] use tree like topologies in which robots can only communicate with or through those above or below them. This results in a system that is highly sensitive to individual failures. Dudek [91] proposes a dynamic topology formation system, which offers a more flexible and adaptable communication system as well as reduces the level of brittleness in the system. Here, communication chains are formed between individuals in a group. The first task of an individual is to join a communication chain. Using only local communication an individual must locate another robot or linked chain of robots and join them. Communication is passed up and down this chain from neighbour to neighbour. If at any time an individual in the chain breaks down it can be quickly detected and the chain either breaks into two or bypasses that individual (if communication can be established between members on either side of the faulty individual). The use of this method also offers the ability to increase the communication range of the system. What is more, although the communication topology is linear the physical topology of the individuals can vary.

Various communication protocols have been considered. Wang [102] uses a media access protocol (similar to the Ethernet protocol). Lin et al [103] investigate the properties of two finite state machine based communication protocols in achieving an object sorting task. The protocols govern co-ordination (through negotiation) and local knowledge exchange. The communication system is integrated into the architecture of the robots and allows for broadcast or peer-to-peer communication over global range. A message queuing system is used to ensure that all messages received are processed. Asama et al [104] use a simple but formal and structured communication protocol. Here communication is used as a backup system to a rule based planner. Its role is to resolve deadlock or collision situations through priority based negotiation.

Dudek et al [91] state that in order for the return on multi-robot environments to be maximised the robots should be well organised and co-ordinated. To achieve this they require high levels of inter-robot communication, which assumes that there is no cost associated with the act of communication. In reality, there invariably is, be it in the form of increased energy consumption or timeliness and agility due to delays caused by competition for the communication resources. In such cases the communication requirements of a task can be reduced substantially by giving individuals the ability to model the intentions, actions and states of others, allowing them to some degree predict the motions of others. Fukuda et al [105] investigate this avenue as a way of reducing communication in their CEBOT environments.

Parker [106] performed various tasks with collective robots with and without communication and found that global awareness of the state of others improves task efficiency. Yoshida et al [107] showed that for systems using local communication there exists an optimum group size as well as communication delay after which the use of communication is detrimental or misleading. Fakuda et al [108] show that in the use of distributed knowledge acquisition within an environment containing autonomous robot cells (CEBOT environment) there exists an optimum communication range, beyond and before which the performance of the system degrades. Works by Akin [109,110]

investigated the effect of group size and different types of communication on a groups ability to perform a set of tasks. This showed that there was on optimal group size and that the communication of state information can be beneficial in some tasks.

The works in the previous paragraph all highlight the fact that the communication needs of a system are very tightly linked to its function and nature. As a result communication cannot just be thrown into a system haphazardly, there needs to be a tailored approach to the communication system. Some work to this end has been undertaken by Yanco and Stein [111] using supervised learning techniques, in which a common and adaptive communication protocol emerged between a group of mobile robots working co-operatively. Their approach allowed the robots to define an appropriate level of global communication for the given task through a reinforcement based learning system. The convergence of the system was, however, exponential on the number of actions and signals. Evolutionary approaches offer the potential to produce systems with either tightly or loosely coupled communication systems in an unsupervised manner. A review of the current literature pertaining to this is presented in the next section.

### 3.3.2 Application of evolution to communication and co-ordination.

There are 2 main approaches through which controllers for mobile robots can evolve, as an explicit control program [6] or as some form of circuit structure [7]. The first form tends to be based around some form of behavioural language, which typically produces programs using the genetic programming paradigm. The latter form includes structures such as neural networks or electronic schematics. The categories are further sub-divided dependent on the level of the evolutionary material used. These can either be simple low-level primitives or high-level functional routines. The proclaimed benefits of using simple primitives is that they offer greater scope for ingenuity on the part of the evolutionary process in contrast to the restrictive nature of high-level constructs. However, having such freedom has a tendency for prohibitively large search times before any meaningful high-level behaviour emerges. These large search times are avoided by using high-level functions which also help eliminate difficulties in analysing

the behaviour of the evolved controllers, which arise through the use of low-level based controllers. The high-level functions are usually expressed as a set of behaviours, which via incremental learning can be used as building blocks to other behaviours, which lead to the goal repertoire being achieved. However, perhaps the greatest perceived drawback of high-level functions is the selection and crafting of the functions, which is not formalised and is time consuming.

Pearce et al [112] and Ram et al [113] use GAs to evolve the control parameters of schema based reactive control systems for navigation of autonomous robots. Menczer and Belew [114] use GAs to evolve neural based sensors for robots. Colombetti and Dorigo [115] use a classifier system based on a distributed GA to evolve behaviour-based controllers for robots. To speed up the convergence process, incremental learning through shaping is used. Basic behaviours are first learnt followed by their co-ordination. Miglino et al [116] use GAs to evolve a neural net based controller for a mobile robot, performing a wandering task within a small grid arena. Lund [117] also used GAs to evolve neural net controllers but applied them to the task of navigation and obstacle avoidance using active perception. Both these latter works used hybrid evaluation (part simulation and part physical) processes as part of their controller evaluation. Floreano et al [118] evolve neural net based controllers on physical robots, whereby they show that it is possible for the evolutionary process to define and implement subtasks not specified in the fitness function but which are essential to the completion of the task. As well as this they also show that the robots also evolve an internal representation of the environment. However none of these works incorporate communication as part of the control systems of the robots.

It is generally argued that for communication to be adopted it should be an aid to significant advances, so in evolutionary terms its use must offer some form of survival benefit. The current approach to this is to embed information within individuals in the environment. This information is essential to the task and can only be accessed or shared via communication.

Work concerning the ability of the evolutionary process to evolve communication in simulated worlds has been conducted by MacLennan et al [119,120], Werner et al [121,122] and Noble et al [123]. These works show that evolutionary approaches allow for the emergence of communicating agents in tasks where individual agents have access to internal information required by others in order to accomplish the task at hand. However the environments in which the agents are evolved are very abstract, lacking any notion of proximity, geometry or time. Another drawback of these methods is their dependence on state transition tables to represent the rules of communication. This method leads to the ambiguous use of symbols in communication and unduly large search times. Work by Reynolds [124] and Qureshi [125] tackle a similar problem but use simulated mobile robots to confirm the potential of the evolutionary process to utilise communication to solve problems. The communication used in all these cases is global. Although work has been done applying GAs to the notion of evolving a use for communication the majority of it concentrates on using genetic programming (Oliphant [126] uses GAs to evolve a very simple form of communication ,known as Saussurean communication, between individuals. However, this also suffers from the drawback of requiring state transition-like structures).

Andre [127] evolved map making programs for a simulated robot and Handley [128] also evolved plans for a simulated robot which were represented as a program. Crosbie and Spafford [129,130] show that intrusion detection systems for a computer system based on software agents can be evolved. Spencer [131] uses a GP to generate efficient programs which allow a six-legged insect to walk about in a simulated environment. Sims [132 - 134] evolves the morphology of virtual creatures in 3-d environments along with their neural control systems. The system produces various locomotion strategies, which are competitive and individualistic. Reynolds [135] evolves vision-based herding behaviours for robots. Reynolds [29,28] also evolves programs for obstacle avoidance and corridor following. All these works are concerned with the task of developing or co-ordinating agents. However, none of them investigates the effect communication will have on the systems. This problem was first tackled in the seminal papers of Werner and Dyer [122] and MacLennan and Burghardt [136]. Werners and Dyer evolved

communicating agents as an aid to mating. In their experiments, the males were blind but mobile and the females were sighted but immobile. MacLennan and Burghardt's work was concerned with co-operative communication. The simulated organisms they use in their experiments have private local state information available to them and they also have access to the global state of the environment. The organisms have only two classes of behaviour open to them: emit a signal or act in response to a signal. A better fitness is achieved by responding to a signal with an action that matches the signaller's local environment state. As well as showing that communicating agents evolved the authors also showed that the use of learning rules improves the effectiveness of communication. However the communication evolved was in some cases ambiguous. The works of Haynes et al [137 to 140] are concerned with producing programs which facilitate co-operation between agents. The predator-prey pursuit problem is used to investigate this. A single prey is pursued by a group of four predators. They state that by giving all the predators the same program (i.e. a homogeneous system) the population of programs represents an implicit co-operation strategy to capture the prey. Only horizontal and vertical movements are permitted and all motions are carried out at the same time. Co-operation strategies are produced without the need for explicit communication between predators or prey. They also found that the use of heterogeneous agents reduced the cases of deadlock as compared to homogeneous cases, which suggest that the level of implicit co-operation in homogeneous systems varies dependent on the situation. Haynes [141] tackles adaptive agents in hostile environments. Agents which exist in a simulated world comprising of energy cells, mine cells, free cells and other agents, having a restricted level of communication. They can only communicate things they have found out for themselves with agents which share the same cell. Programs are evolved which allow the agents to move around the environment avoiding mines and collecting energy. Di Paolo [142,143] evolved communication between static agents involved in games of co-ordination. A more general definition of communication was used here, one which involved observing behaviour, i.e. sensor based communication, as well as information transfer. The aim here was for agents to survive and reproduce, the main driving force for this was food consumption. Di Paolo [142] found that clusters of agents were formed and that

communications took place within these clusters. Further if the cost of communication was made detrimental to individuals (by increasing the food reward to co-operating agents) it was found that communicating agents were still produced. Some of this communication tended to be deceptive i.e. deliberately communicating the wrong information but the key factor in determining whether communication was used was the spatial relationships between individual agents as well as the distribution of food within the clusters. The reason for this can be seen from the following example. If there is more food on the edge of a cluster than in the centre it makes sense for those agents in the centre to communicate with those on the edge in a non-deceptive manner. Although by using deceptive communication an agent could maximise its local return it would suffer once all its local resources had been consumed. As a result of this the use of deceptive communication tended to be a short-lived strategy, being replaced by more co-operative communication. Di Paolo also questioned the need for internal information for the success of evolutionary approaches to communication. De Bourcier and Wheeler [144] considered the role reliability of communicated information plays within a society of artificial autonomous agents. These agents evolved signalling and receiving tactics in order to effectively compete for food. Their results showed that reliable sources of information about an individuals abilities are preferred to unreliable ones (or deceitfulness). However, if the cost of obtaining this reliable information is too costly then more attention will be paid to unreliable signals. Fyfe and Livingstone [145] show that the aptitude for a common language favours the evolution of a common representation capability. Here individual agents were allowed to evolve their own internal representation of features in the environment and then try to communicate with other agents about these features. As well as this the structure of the neural net which was used for representing the environment was concurrently evolved. So although the individuals had different representations of the environment they all had a common representation capability. Billard and Dautenhalm [146] were also concerned with learning features and establishing communication about them. Here however the communication about features is used to convey to separated robots the location of another. In this work a teacher robot is followed by a pupil to whom he communicates words about features in the environment as they are encountered. The pupil learns to

associate its sensor readings with these words and locations so that if it is separated from the teacher it can use the teachers communications as additional information to aid finding him.

Steels [147..149] showed that it is possible for a common language to emerge between agents engaged in language games. These games are either concerned with naming, spatial descriptions or feature discrimination. Self-organisation is used as opposed to a genetic based evolutionary method. This method places stronger emphasis on the communication of agents. Vocabularies emerge between a distributed group of agents which support polysemy, synonymy and ambiguity. These vocabularies are robust and can be quickly acquired by new agents joining the group. This method is based on the hypothesis that language evolves within the lifetime of the individuals and hence can be considered dynamic, whereas in the simple languages of MacLennan et al [136,119,120] and Wernern and Dyer [122] are constant. The form of communication used by Steels was also global, but on a one-to-one basis. The agents do not form totally identical languages but have sufficient in common for the global system to achieve sustainable communicative success. Maede et al [150] extend the work of Steels to consider language contact. Here they investigate what happens when two separately evolved languages meet up. They found that the dominant language prevailed although some borrowing from the other language did occur when unfamiliar cases occurred.

### 3.3.3 Robot communication and co-ordination review summary

While the literature review addressed the issue of evolving communicating agents, the questions of when and what they should communicate have not been addressed. The work presented in this thesis will focus on these questions in relation to evolving co-ordination amongst multiple robots. It seeks to establish how the form of communication available to an evolutionary based method affects its ability to utilise communication in order to solve a given task. The work will adopt a mixture of high- and low-level approaches to demonstrate the ability of the evolutionary process to

integrate communication into evolved control programs allowing for more flexible forms of communication.

The majority of the work undertaken which is relevant to this area concerns either the pursuit-evasion problem or the rendezvous of individuals. To establish a base case, this approach is adopted and applied to the general problem of robots meeting up using a low-level approach, which will then be extended to see if the choice of when and how often to communicate can be determined (using a high-level approach). Finally, the question of deciding when and how to use communication to control directly the behaviours of other robots both individually and collectively is investigated.

## 3.4 In summary

This chapter has reviewed the literature related to path planning for mobile robots. Whilst solutions of varying sophistication and complexity were identified each suffers from some of the following limitations:-

Potential field.

- Existence of multiple local minima, resulting in null driving forces.
- The inability to guarantee convergence to goal in the presence of local minima.
- The difficulty of generating globally optimal paths.

Configuration space

- High computational demand for mapping between real-world and C-space.
- Long search times due to large dimensionality of model.
- Approximation errors leading to the failure to find paths when they exist.

With this background, the work of this thesis investigates the use of GAs for path planning. In particular the following points are addressed:

- How best to represent the environment.
- How best to represent the paths.
- To identify the key features of the path planning process which can serve to guide the evolutionary process towards effective and robust solutions.

In addition, this chapter has reviewed the literature related to communication in multiple robot (agent) systems. It has been identified that the questions of when and what information should be communicated have not been addressed. This question will also be addressed in this thesis.

# Chapter 4

# TASK I: PATH PLANNING

## 4.1 INTRODUCTION

In the light of the shortcomings of the path planning methods highlighted in chapter 2, the issue of evolutionary mobile robot path planning is selected as the first topic of research, the aim of which is to develop an understanding of the requirements, abilities and performance of a genetic based path planner in static and known environments. This problem must be formulated in a way that is suited to solution by a GA. Also the characteristics of the representations most appropriate for GA must be identified and techniques developed to boost the effectiveness of the planner where any drawbacks exist or any significant performance enhancements can be gained.

The formulation of path planning for solution by genetic algorithm is therefore concerned with identifying:

1. The appropriate representation of the environment.
2. The appropriate representation for the paths.
3. Identifying key features of the path planning process which can serve to guide the evolutionary process toward effective and robust solutions.

These points are addressed by performing experiments with a single mobile robot, whose path must be planned in an environment which is occupied with static obstacles.

## 4.2 IMPLEMENTATION OF PROBLEM

### 4.2.1 Preparatory experiments

Associated with the use of some systems are sets of what could be seen as possibly artificial or arbitrary settings (or magic numbers) which play a significant role in the functioning of the systems. This often therefore leads to the questioning of the sensitivity of the system or robustness of the solutions it produces to these settings. However, in many of these cases there is usually either a commonly used group of settings or a set of general principles governing the selection of settings. In evolutionary systems, the presence of magic numbers is quite pronounced, which is to some degree a result of the way they function and it offers them greater scope for fine tuning performance and for domain independence. Due to the robustness of these systems, they tend to be quite forgiving when it comes to the settings of these parameters. This work, since it is based around the use of evolutionary tools, contains a number of such parameters. In the most part their values were determined by running a set of preparatory experiments to determine the associated effective values for these parameters. The exact nature and reasoning behind each choice will be considered at the relevant points in the thesis. Those used in this particular chapter and their associated settings are given below:

- Maximum number of generations. 300
- Population size. 130
- Reproduction preservation size. 16
- Maximum chromosome length. 224
- Mutation rate 0.001
- Resource allocation percentage. [3, 26, 16, 8, 8, 16, 23]
- Avoidance flow constants. $Flow_{min}$ 10, $K_{HOR}$ 2, $K_{VERT}$ 1
- nDynamic step size limit. 8
- nFixed step size value. 3

- Number of independent test runs. 16

The termination criteria for the path planning process was the number of elapsed generations. The maximum possible number of generations was 300. This level was determined as a result of preparatory experiments, which showed that when an explorative crossover is used the first collision-free paths were typically found no later than generation 260. The experiments also indicated that the most noticeable amount of path optimisation takes place within the first 20 to 30 generations of a path being found. An additional 40 generations was added to this late find value, giving 300, to allow for some optimisation of the paths found in these later stages.

The compiler used placed an arbitrary limit on data structure and segment sizes, so this in turn effected the permissible set of population sizes. Since two populations need to be present at any time this placed further restriction on the size of an individual population. Under these restrictions the largest possible population was 250. With this upper limit a set of experiments was conducted to see the effect various different population sizes had, the sizes used were 50, 80, 130, 180, 250. These experiments showed that the gain from 130 to 250 was not in proportion to the increase in processing time, also that the population size offered less benefit than the use of a highly explorative crossover operator. Since processing time is a key factor, a small value would be preferred, however for diversity and efficiency reasons a large population size is preferred, as a result 130 was chosen.

The reproduction method used here is elitist and requires a number of individuals from the current population to be copied into the next generation. The setting of this value is governed by the fact that typically 10% of the population does not undergo crossover. Here this is achieved by using an overlapping population. Experiments showed that a level slightly higher than this 10% level, one of 12.6% (or 16 individuals using a population size of 130) offered the best performance. This level allowed for an effective balance between preservation and exploration within the GA to be established.

The maximum length of a path (i.e. Chromosome) is set at 224. Due to the nature of the path representation used here, the dimensions of the environment have a relatively minimal effect on this length. What is of most significance is the deceptiveness of the path planning case. In these cases the resultant path must take a roundabout route in order to get to its destination. As such, the length of the chromosome should be enough to handle most of these cases. Owing to the fact that the length of the chromosome impacts on the processing time as well, this should also be kept as small as possible. Experiments showed that for the obstacle distribution and the path planning cases used here a chromosome length of 224 was found to be more than adequate.

The mutation rate was set at the standard level, 0.001, because early experiments showed the main genetic operator was crossover.

Experimentation found that various resource allocation percentages allowed paths to be generated. The settings [3, 26, 16, 8, 8, 16, 23] reflect the general processing requirements needed for each constraint, indicating its ease of accomplishment or its relative importance.

The constants used in the avoidance flow algorithm were $Flow_{min}$=10, $K_{HOR}$=2; $K_{VERT}$=1. The main factor governing these levels was avoiding the possibility of numeric overflow. The $K_{HOR}$ and $K_{VERT}$ levels were also chosen to minimise mathematical calculation, while still allowing the benefits of the avoidance method to be gained. The $Flow_{min}$ level was also chosen such as to reduce the chances of the summed flow values causing an overflow, whilst at the same time marking a distinct barrier between obstacles and free space These levels where verified as effective through experiments (see section 4.2.5 for more operational detail).

Various path representations are investigated in section 4.3.2 one of these is nDynamic, associated with it is an upper step size limit which is set to 8. This level was chosen such that it made the representation distinguishable from others and also so that the

characteristics of the path representation could be made more apparent (section 4.3.2 gives more detail on path representation).

Various path representations are investigated in section 4.3.2; one of these is nFixed, associated with which is a constant step size, which is set to 3. Experiments showed that there was a limited set of possible values for the step size of this representation (section 4.3.2.1 gives more details). This value was chosen because its use distinguished it significantly from the other representation used.

The number of independent test runs used here is 16, which was the largest number of runs time permitted to be used. Using this number of runs the time taken to evaluate the overall performance for a single test would take between 30 and 45 minutes. Further, considering that each test was also to be performed again under a different operator configuration meant this time would increase two or three fold. Also, a number of different tests were also to be run giving an overall time requirement of 540 to 810 minutes (or 8.5 to 13.5 hours). However experiments showed that this number of runs was sufficient to highlight the variability between results produced by an individual configuration in each of the given tests.

### 4.2.2 Environment representation

A path planning technique, which would be applicable to environments with arbitrarily shaped obstacles, is desired. To accomplish this a two-phase environment representation has been devised. First a discretised bit map, from which information on the obstacles is extracted, is used. This takes the form of identifying which points in the environment are occupied and to which obstacle they belong. The second representation is generated using the extracted obstacle information. The environment is represented as an integer map where the points in each obstacle are assigned positive non-zero values indicating their distance from the avoidance side or sides of the obstacle (see section 4.3.5 for more details.) Any unoccupied points in the environment are given zero values.

### 4.2.3 Path representation

A path representation is required which can be manipulated easily and which will reduce the burden of the GA's search task. A displacement path structure approach has been devised, which reduces the GA's search space by removing the possibility of producing discontinuous paths. Such a structure indicates the route the path will follow by a movement direction relative to (or displaced from) the current position. The possible directions of movement are limited to eight and correspond to the directions of a compass. This is an easily representable structure, requiring fixed-length one-dimensional arrays where each element of the array indicates a direction of motion. The directions are coded into chromosomes as values (genes) ranging from 0 to 7 (see Figure 4.1) giving a gene length of 3 bits. The fixed length nature of the array defines an upper limit on the path length but the fitness function allows it to vary in length up to this limit.

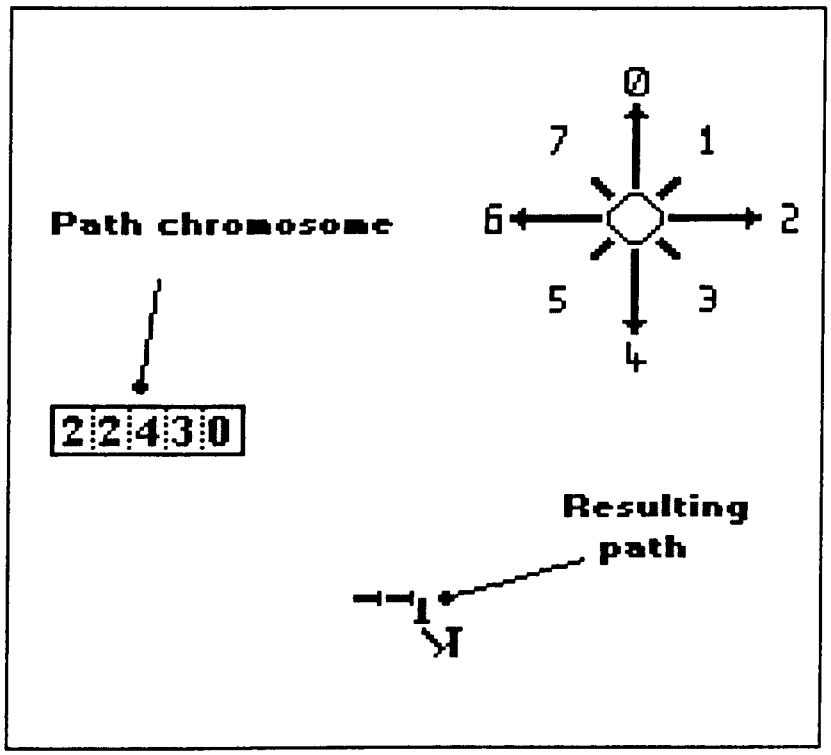



Figure 4.1 An example of the relative path representation used by the GA-planner.

### 4.2.4 Definition of path planning constraints

Evolutionary systems require some metric to define the performance of its individuals (a fitness function). This implies that the system being evolved can be expressed as a set of constraints, which can be evaluated. For the purpose of path planning the constraints used should encapsulate those properties which lead to a collision-free path being found and allow for their optimisation. To this end seven constraints were identified (all of which have to be minimised) as being relevant and prioritised. These can be seen below in descending order of priority:

i) Number of points in the path outside the workspace.
ii) The sum of the offsets of the closest point to the goal along the x and y axis.
iii) The sum of the associated parameter of the points in ii).
iv) Number of points in the path, which collide with obstacles.
v) Avoidance sum of colliding points in the path.
vi) The excessiveness of the path length.
vii) Number of direction changes in the path.

Constraint *i)* is required to allow the GA to reward paths for staying within the bounds of the environment. Some early exploratory experiments identified the need for and the grouping and ordering of constraints *ii), iii), iv)* and *v*). Two distinct groups were formed, constraints ii) & iii) and constraints iv) & v). The first grouping is required to enable the GA to navigate the path to the goal. The division of these constraints is required to enable the path to approach the goal along the axis of least resistance. The second grouping is required to aid collision avoidance. Since the ordering of the constraints allows for the path to pass through obstacles in order to get to the goal, a mechanism is required to allow the GA to reward those paths which slide off the obstacles, finally avoiding them altogether. To achieve this collision avoidance, counting alone is not sufficient (constraint *iv),*) some additional avoidance information is needed. This is offered by constraint *v)* in the form of

the sum of the avoidance distances associated with each obstacle point. Constraint *vi)* is responsible for optimising the length of the path while constraint *vii)* smoothes it out.

The percentage distribution of genetic processing resources between the constraints is given in table 4.1. It should be borne in mind that the preservation size is 16, which is (130-16) giving an effective population size of 114:

| **Constraint.** | i | ii | iii | iv | v | vi | vii |
|---|---|---|---|---|---|---|---|
| **Resource allocation (%).** | 3 | 26 | 16 | 8 | 8 | 16 | 23 |

Table 4.1 Percentage of population allocated to each constraint for reproduction.

### 4.2.5 Obstacle avoidance method

A method to overcome the problem of lack of collision count discrimination was required. In other words if two or more paths collide with an obstacle the same number of times, what order should they be evaluated in. In such cases some path collisions are close to the edge of the obstacle and others are further away. In order to reward those paths which are close to the edge of the obstacle and thereby encourage obstacle avoidance, an obstacle colouring method called 'Directed Flow' is proposed. This requires that firstly a global set of avoidance edges be identified. These are the sides of the obstacle to which a colliding path will be drawn, which can be either: left, right, bottom, top or some combination of the four. These avoidance sides must be identified prior to path planning taking into account the position of obstacles relative to boundary walls. The directed flow value is calculated for each point in an obstacle. It indicates the distance of that particular point from its designated avoidance side or sides in the obstacle that is, the path is encouraged to avoid an obstacle by moving along a particular side. So, an environment using top_left obstacle avoidance will always encourage the GA to avoid it by passing around the top and left edges of an obstacle. An example of colourings of some obstacles using top_left avoidance can be seen in Figure 4.2. Obstacle 1 is a solid rectangle with dimensions of 4 x 3 pixels

and obstacle 2 a 3 x 3 pixels cross. Top_Left avoidance is adopted in this work. The algorithm for producing these colourings is given below.

**BEGIN**

Horizontal_Flow_Gradient=$K_{HOR}$

Vertical_Flow_Gradient=$K_{VERT}$

**FOR** I := 0 **TO** Obstacle_Row_Max **STEP 1**

**FOR** j := 0 **TO** Obstacle_Column_Max **STEP 1**

Obstacle[i,j] := $Flow_{MIN}$ + i*$K_{HOR}$ + j*$K_{VERT}$

**END**

**END**

**END**

The role of the minimum flow constant $Flow_{Min}$ is twofold. Firstly, to provide a reference to which the other flow values can be calculated and, secondly, to define the gradient value between free space points and points on the edge of the obstacle. All points that are not part of an obstacle are given zero flow values, so the greater the difference between these points and the edge of an obstacle, the more attractive the free space points will be to the GA. The minimum flow value should be greater than both the horizontal and vertical flow gradients. In the example shown in figure 4.2 the following values are defined: $Flow_{MIN}$=10; $K_{HOR}$=2; $K_{VERT}$ =1;

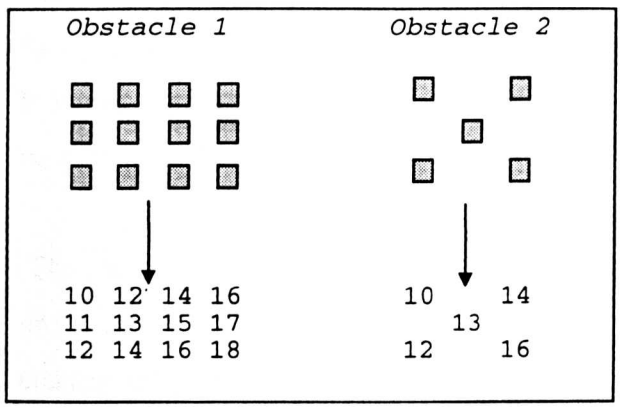



Figure 4.2 Resulting appearance of two obstacles after directed flow avoidance coloring has been applied to them.

### 4.2.6 Implementing the genetic algorithm

The implementation of the aforementioned environment and path representations is straightforward. A two-dimensional array is used to represent the environment, whose size is 80x80 units, and an array used for the path representation. Before this can be applied, the obstacle colouring takes place defining the environment for the GA to process. Preparatory experiments identified a GA with the following configuration:

- Population size 130.
- 300 generations of evolution.
- A chromosome length of 224.
- Uniform crossover.
- Uniform mutation.
- Elitist reproduction, with a preservation size of 16.

In evolutionary terms a small population of potential solutions was used. This was to minimise the amount of time each generation would take to evaluate. The value 130 gave an acceptable time to evaluate and was also large enough to avoid premature convergence. The large runtime (300 generations) is required to give the GA sufficient time in which to produce collision-free paths. The chromosome length is set high allowing for longer paths than necessary, giving the GA more freedom to find a collision-free path. The use of one or two point crossover operators resulted in premature convergence to local minima. However, the use of uniform crossover reduced this likelihood and this, in conjunction with an elitist reproduction strategy, offers the most potential.

The standard GA, where fitness is a single point value can only solve single constraint problems. However, the need was previously identified for multiple constraints in order to convey to the GA the nuances of path planning. This need for multiple constraints and the use of elitist reproduction (in which the best n paths of each generation must be identified) requires the development of a different representation and ordering mechanism for

chromosome fitness values. The multiple constraints could be reduced to a scalar constraint by using some kind of sum or weighted sum of the individual constraint values. However, this eliminates the ability to convey to the GA the order of constraint optimisation. Instead, representing the fitness value as a vector, where each element of the vector corresponds to a constraint, preserves the multiple nature of the constraint. The use of this vector representation makes the ordering of the fitness more complicated. This is resolved by ordering the fitness vector elements in order of decreasing priority from left to right and employing a concept of dominance. This allows for the most dominant fitness vectors to be identified. A fitness vector is considered dominant over another if at the first differing vector value (starting from the left) its value is the larger. However, since we are minimising the constraints, what we require is the least dominant fitness vector i.e. during the comparison of the differing values, the lower valued vector is selected. With this, the fitness vectors can be appropriately ordered. The need to preserve the prioritisation information to guide the GA in its search has been previously mentioned. This combination of prioritised vectors and elitist reproduction offers a further benefit. If at any point in the search process the high priority constraint reaches a plateau then the next highest constraint temporarily gets to guide the search process until the previous high escapes. This said, whilst the highest priority constraint is guiding the main thrust of the search, the lower ones are still being optimised and contributing information to the search process. This property is made possible by the reproduction process. Since multiple constraints are used, during the reproduction stage of the GA cycle each constraint gets to contribute a number of offspring to the next generation based on their performance on that particular constraint.

## 4.3 EXPERIMENTS AND ANALYSIS

### 4.3.1 Experiment I

*Aim:* To establish whether the proposed configuration for a genetic algorithm based path planner has any path generating ability.

In this experiment the GA-planner was used with the configuration described in section 4.2. Its application was initially in an obstacle free environment. This is the simplest path planning case, so if the GA-planner fails in this then we can conclude that it is very unlikely it would succeed in more sophisticated obstacle ridden environments and as such the GA is not a suitable tool for path planning. However, if it succeeds we have demonstrated that it has basic path planning abilities and merits further investigation. Four path planning cases were tested with in the empty environment, each one having distinct starting and ending locations (which can be seen in table 4.2) with the aim of highlighting any weak points in the planner or its constraint system. Dependent on its performance here a further set of tests in environments containing obstacles were available, in which its performance in cluttered, highly cluttered and deceptive environments would be ascertained (these can be seen in table 4.3, scenarios 1 to 3 used here can be seen in figures 4.11 to 4.13). For each of the test cases the GA was run 16 times, with each run having a different set of random seeds for the genetic operators. These test where all carried out on a HP 700 series workstation run under UNIX. The GA code was implemented in C++ (code included in desk 1 appendix C).

| TEST ID | START LOCATION | END LOCATION | OPTIMAL PATH LENGTH | OPTIMAL PATH SMOOTHNESS |
|---|---|---|---|---|
| A1 | (0,0) | (79,79) | 80 | 0 |
| A2 | (0,79) | (79,0) | 80 | 0 |
| A3 | (10,20) | (10,50) | 30 | 0 |
| A4 | (25,8) | (35,40) | 34 | 2 |

Table 4.2 List of test attributes for empty environment test cases.

| TEST ID | START LOCATION | END LOCATION | SCENARIO | OPTIMAL PATH LENGTH | OPTIMAL PATH SMOOTHNESS |
|---|---|---|---|---|---|
| B1 | (79,0) | (79,79) | 1 | 121 | 7 |
| B2 | (79,79) | (0,0) | 2 | 91 | 3 |
| B3 | (65,5) | (4,70) | 3 | 70 | 4 |

Table 4.3 List of test attributes for initial cluttered environment test cases.

#### 4.3.1.1 Results from experiment I

The time taken to process each generation varied depending on the lengths of the paths the chromosomes produced. The process employed in converting the chromosome representation to the corresponding path brought about this variation in time. This required that the chromosome be converted to a path one gene at a time and that when any point in the path passed within a fixed distance of the goal point the conversion was halted and the path was considered as having reached the goal. This process was incorporated as a way to speed up processing time. Although a time per generation cannot be given, the average time taken for a complete run gives a better estimate to the general processing requirement of the algorithm. The average time taken for a complete run of 300 generations was 95 seconds. From this, the average time taken to find the first collision-free path can be determined. Although the runtime was more or less the same regardless of environment, the time taken to find the first collision-free path varied depending on the obstacle distribution in the environment. In obstacle free environments the average time taken was around 12 seconds and in the case of cluttered environments 29 seconds.

The four planning cases for the obstacle free environments can be seen in figures 4.3 to 4.6. These figures show the start and end points of the proposed path as well as the optimum solution. They are given as a visual guide to the effectiveness of the paths

produced by the GA path planner. Sections of paths in grey show where the optimal straight-line path diverges from the optimal disrcetised path. The number of generations the GA-planner took to find the first path between the start and end points varied in each test case. The variation was dependent on the distance and inclination between the start and end points of the proposed paths. However, the average time taken was about 38 generations. Paths whose inclination was not a multiple of 45° took comparably longer to produce and were less smooth. This can be attributed to the discretisation and restricted movement bias in the environment and path representations. For any non-multiple inclination, a number of additional path motions is required to compensate for this factor. The initial paths generated were very haphazard and took complicated, roundabout routes to get from the start to the goal. An exception to this tended to be a paths whose optimal length was less than about 9 units. In these cases, the initial paths the GA-planner produced tended to be near optimal. The haphazardness of the longer paths was measured using two factors:

- the difference in length between planner's path and the optimal solution (excessive path length,)
- the number of direction changes in the path (smoothness).

The rate at which this haphazardness decreased was an indication of the optimising abilities of the GA-planner. This optimisation of paths was again dependent on the distance between the start and end point of a path and also its inclination, with longer paths taking the most time to optimise and paths with non 45° inclination multiples requiring greatest effort. Only lines inclined at 45° or 90° to the horizontal or vertical can produce straight lines in discrete spaces, all other inclinations result in straight-line approximations, which give a jagged edge to the line. It is this jaggedness which results in excessive direction changes in path. An indication of this feature is that although the length of the line is optimal its smoothness is not. The average time taken to find its optimum path in a straightforward case (path not too long in distance and an inclination angle which is a multiple of 45°) was about 27 generations. However, in non-straightforward cases this increased to around 45 generations. In all cases, the optimum straight-line path was rarely produced.

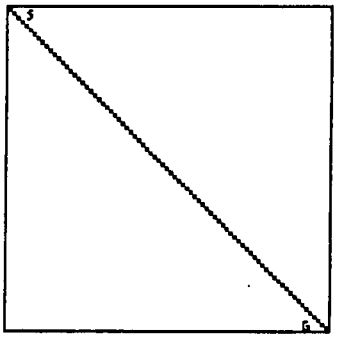


Figure 4.3 Test case A1 with optimal path.

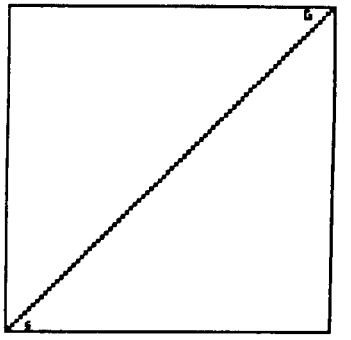


Figure 4.4 Test case A2 with optimal path.

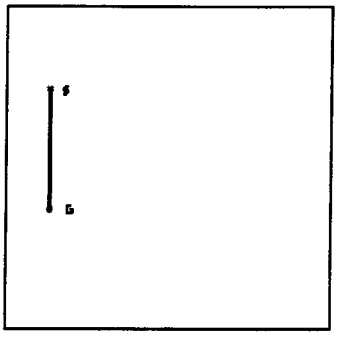


Figure 4.5 Test case A3 with optimal path.

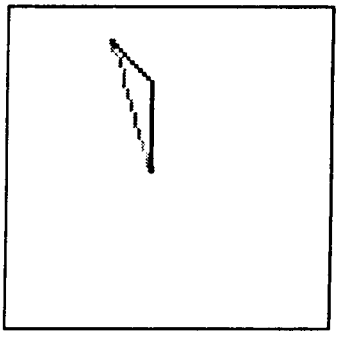


Figure 4.6 Test case A4 with optimal path.

Although the GA-planner was capable of generating optimised paths between two points, a problem did exist in the system. Although all the paths were classified as valid, collision-free and joined (those reaching the goal without passing through obstacles and staying wholly within the environment), the question was did they derive from paths which had been invalid, i.e. a path which reached the goal and avoided obstacles by having one or more of its points pass outside the environment. This situation was relatively common, particularly in cases where the start and the end point in the path were close to the edge of the environment, which indicates a possible weakness in the specification of constraint i), which is meant to reduce the effect such paths have. In order to remove this possibility and

to improve processing time a repair operator was implemented to cater for such situations. The job of the operator was to ensure that all chromosomes produce valid paths. This it achieved by folding any section of a path, which exceeded the boundary of the environment back on itself. The implementation of this operator reduces the possible number of invalid paths and as such stops the GA-planner wasting processing time on invalid paths but it also increases the path search space, allowing for multiple representations of the same path. The repair operator could act in one of two ways:

- re-interpret the chromosome leaving it unchanged (*repair-1*),
- re-interpret the chromosome and replacing it with the re-interpretation (*repair-2*).

The experiments were re-run using both these operators in turn and the time taken to find the first collision-free path used as metric (since there are no obstacles, this effectively is the first path to reach the goal point). In both cases the evolutionary process was slower, however the use of the *repair-1* operator was found to offer faster and more consistent performance than the *repair-2* case. In addition, the average time taken to produce the best path improved in both cases. This was due to the lack of wasted time spent processing paths which did not reach the goal. For *repair-1* this was 15 for straightforward cases and 32 for non-straight cases. As a result *repair-1* was incorporated as part of the standard GA-planner configuration. Some examples of the paths produced by the GA-planner in these four test cases can be seen in figures 4.7 to 4.10. Tables 4.4 to 4.6 contain information on the performance of the GA-planner in each of its configurations for the empty environment test suit. Table 4.4 shows the average time taken to find the first collision-free path. Tables 4.5 and 4.6 show the average excess path length and erraticness respectively of the best path each configuration produced. Following on from the success in these test cases the GA-planner was given the more taxing task of producing collision-free paths in obstacle filled environments.

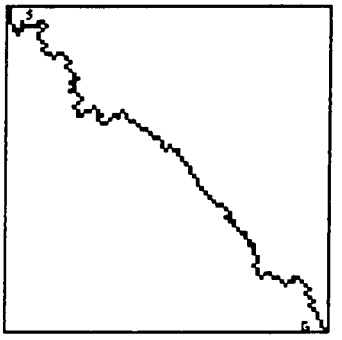


Figure 4.7 Path produced in test A1.

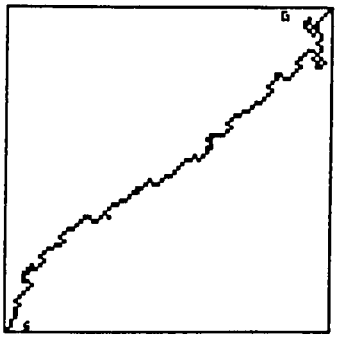


Figure 4.8 Path produced in test A2.

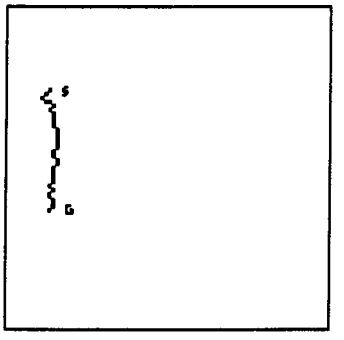


Figure 4.9 Path produced in test A3.

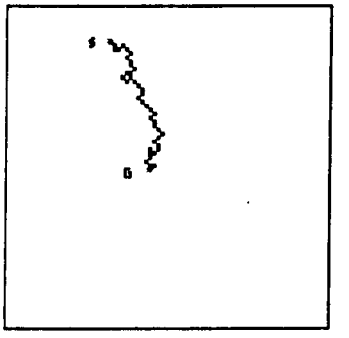


Figure 4.10 Path produced in test A4.

| | | TESTS | | | |
|---|---|---|---|---|---|
| | | **A1** | **A2** | **A3** | **A4** |
| No repair | Mean time taken to find first collision-free path (in generations). | 54 | 53 | 15 | 27 |
| | Standard deviation. | 10.6478 | 9.8146 | 5.7892 | 6.1029 |
| Repair_1 | Mean time taken to find first collision-free path (in generations). | 65 | 63 | 19 | 32 |
| | Standard deviation. | 9.1784 | 9.3517 | 4.3123 | 5.8234 |
| Repair_2 | Mean time taken to find first collision-free path (in generations). | 100 | 76 | 25 | 31 |
| | Standard deviation. | 10.5762 | 10.138 | 5.8631 | 4.9197 |

Table 4.4 Mean time taken to find first collision-free path for various repair-based configurations of GA-planner in obstacle free test cases and their associated standard deviations.

| | | TESTS | | | |
|---|---|---|---|---|---|
| | | **A1** | **A2** | **A3** | **A4** |
| No repair | Mean excess path length. | 60 | 53 | 9 | 38 |
| | Standard deviation. | 12.4713 | 11.183 | 4.4596 | 9.12 |
| Repair_1 | Mean excess path length. | 58 | 47 | 12 | 33 |
| | Standard deviation. | 10.9469 | 19.876 3 | 4.5621 | 8.9862 |
| Repair_2 | Mean excess path length. | 61 | 45 | 13 | 30 |
| | Standard deviation. | 15.4421 | 17.392 | 5.1364 | 8.6237 |

Table 4.5 Mean amount of excessive path length in best paths produced for various repair-based configurations of GA-planner in obstacle free test cases and their associated standard deviations.

| | | TESTS | | | |
|---|---|---|---|---|---|
| | | **A1** | **A2** | **A3** | **A4** |
| No repair | Mean number of excess direction changes in the best path. | 96 | 73 | 32 | 64 |
| | Standard deviation. | 15.7381 | 13.6532 | 10.8719 | 12.8427 |
| Repair_1 | Mean number of excess direction changes in the best path. | 103 | 63 | 28 | 53 |
| | Standard deviation. | 16.8688 | 18.4628 | 10.9003 | 13.7625 |
| Repair_2 | Mean number of excess direction changes in the best path. | 110 | 76 | 30 | 61 |
| | Standard deviation. | 14.8312 | 20.1092 | 11.667 | 14.8094 |

Table 4.6 Mean amount of excessive direction changes in the best path produced for various repair-based configurations of GA-planner in obstacles free test cases and their associated standard deviations.

The three planning cases for the obstacle filled environments can be seen in figures 4.11 to 4.13. These show the start and end position for the proposed paths as well as the associated optimum path through the obstacles in grey. This is for the purpose of visual comparison of the effectiveness of the paths produced by the GA-planner. The presence of obstacles slowed the path planning process down, which was due to the need to avoid them. The time taken to produce the first collision-free path again varied from test to test. However, in the cluttered environments it was comparable, with averages of 37 and 115 generations respectively for the two cases (B2 and B3). The deceptive case took an average of about 110 generations. The general optimisation of the collision-free paths took longer than in the obstacle free case, with the cluttered test cases taking less time than the deceptive one. The average optimisation durations were 75, 103 and 155 generations respectively. Although the optimisation duration was extended, the best paths produced were far from optimum. A sample of some of the paths produced in each of the test cases can be seen in figures 4.14 to

4.16. Information on the average performance of the GA-planner in each of the 3 initial test cases for cluttered environments can be seen in Tables 4.7 to 4.9. Table 4.7 shows the time taken to find the first collision-free path. Tables 4.8 and 4.9 show the excess path length and erraticness respectively of the best paths produced in each of the tests.

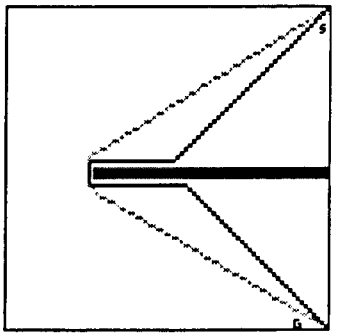


Figure 4.11 Scenario 1 deceptive environment. Optimal solution for test B1.

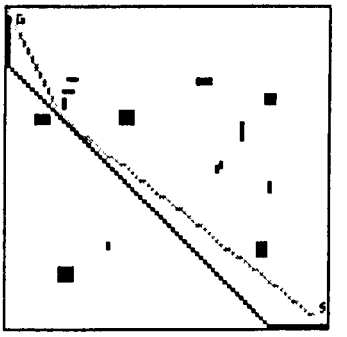


Figure 4.12 Scenario 2 cluttered environment. Optimal solution for test B2.

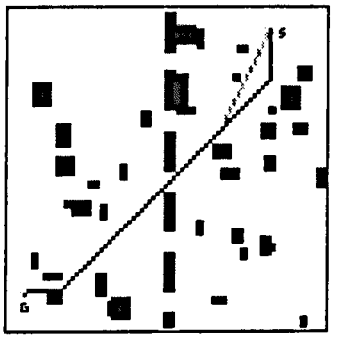


Figure 4.13 Scenario 3 very cluttered environment. Optimal solution for test B3.

| | TESTS | | |
|---|---|---|---|
| | **B1** | **B2** | **B3** |
| Mean time taken to find first collision-free path (in generations). | 110 | 37 | 115 |
| Standard deviation. | 19.87 | 5.4381 | 54.9378 |

Table 4.7 Mean time taken to find first collision-free path in initial obstacle filled test cases and their associated standard deviations.

| | TESTS | | |
|---|---|---|---|
| | **B1** | **B2** | **B3** |
| Mean amount of excess path length in best path (in generations). | 100 | 60 | 85 |
| Standard deviation. | 34.4154 | 24.6041 | 55.3302 |

Table 4.8 Mean amount of excessive path length in best paths produced in initial obstacle filled test cases and their associated standard deviations.

| | TESTS | | |
|---|---|---|---|
| | **B1** | **B2** | **B3** |
| Mean amount of excess direction changes in best path (in generations). | 130 | 96 | 120 |
| Standard deviation. | 31.7038 | 26.1249 | 48.0973 |

Table 4.9 Mean amount of excessive direction changes in the best path produced in initial obstacle filled test cases and their associated standard deviations.

To end these initial experiments the effect the mutation rate had on the path was investigated. It was found that increased or variable mutation rates had a detrimental effect on the time it took to find the first collision-free path. However, they offered a slight improvement on the lengths and erraticness of these paths.

#### 4.3.1.2 Discussion of results from experiment I

The experiments undertaken show that a GA-based path planner can produce collision-free paths between two points in both cluttered and obstacle-free environments. The speed of path production is faster in obstacle-free cases because of the large number of possible routes between the two points. When obstacles are introduced, the planner has to rely heavily on the obstacle avoidance method to produce collision-free paths. The way in which this works by gradually encouraging the path to flow off the obstacle aids significantly in avoiding the obstacles. However, there is a computational overhead associated with it. This means that the planner's speed will vary according to the complexity, size and position of the obstacles in the environment. Although the speed of path production is faster in the obstacle-free case, the optimisation duration is a lot longer. This can possibly be attributed to the extremely large number of possible solutions available and the lack of processing resources allocated to the length and erraticness constraints *vi)* and *vii)*. In this case, the population is swamped with distinct collision-free solutions and the processing resources are set to ensure this continues. This is fine for the obstacle-filled environments because the number of possible paths between the two end points in a path is much less so the distinctness of each path would be much lower. However, in the obstacle free case it means less time for optimisation since distinct paths are being produced faster than paths can be optimised. A possible way to avoid this situation is to use adaptive genetic resource allocation, so, as the number of distinct collision-free paths in the population reaches some threshold, processing resources are shifted from constraints *i)* through *v)* to constraints *vi)* and *vii)* and should the percentage fall back below this threshold the resources are then shifted back. What this means is that once n% of paths are collision-free forget about producing any more and just concentrate on optimising them and should things deteriorate switch the priority back to the production of collision-free paths. Although this method would have great advantages in the obstacle free case, its application in obstacle filled environments is less certain. It may well be detrimental since the number of collision-free paths can change rapidly from generation to

generation in these cases and such an approach, it would seem would require long periods of stability in the number of collision-free paths for it to be beneficial.

The introduction of the repair operator reduces the processing load on constraint *i)* and increases the optimisation duration as well as reducing the haphazardness of the paths. The chosen repair operator, *repair-1*, succeeds by allowing multiple ways for representing a path as well as increasing the diversity of the population. It is this diversity in the population that reduces the chances of premature convergence. It achieves this diversity by allowing different chromosome representations to exist in the population, which decode to exactly the same path. As well as diversity it also creates additional evolutionary paths for arriving at a given environment path, which makes the search process faster. In order for the operator to work well, it should be incorporated as part of the fitness function, so that it will not have the opportunity to alter the chromosome representation.

The ability to produce paths shows that constraints *ii)* and *iii)* are working, that is, the path guidance system is valid. However, the need for such a major role for these constraints is raised. Should they be responsible for bringing any prospective paths from one end of the environment to the other or should they just be responsible for keeping the paths on or within a designated distance of the goal (at the moment they do both.) The latter will be more productive because it would cut down on processing time. However for such an approach the role of getting the paths themselves to the goal must be done by another operator, a *Join* operator, which would work by generating a straight line path between the two ends of the path then randomly inserting direction changes into the path such that it explores the space within the environment but remains joined to both ends of the path. Such an operator will be used, as part of the initialisation process for the paths of the GA. This operator will make the obstacle free environment a trivial case. But in the obstacle filled environment it will allow more time for optimisation and offer more options for collision-free paths to the goal.

Although the GA-planner finds the initial collision-free paths quite early, the quality of these paths is always very poor due to their haphazard nature. This, as well as the poor levels of final optimisation of the paths, is attributable to the highly flexible nature of the path representation. This flexibility is great when obstacles are all around but is very detrimental in large open spaces (this accounts for the high level of performance of the planner in cluttered environments and the poor performance in obstacle free environments). Since such flexibility increases both search time and search space, a more suitable representation is desirable, notwithstanding the flexibility of the path representation may not be sufficient to overcome highly deceptive problems. The example used here gave the GA a lot of time and space to see the problem was deceptive and to start taking measures to overcome it. However, if the end points in the path were brought much closer to the obstacle there will be less time to react (which is due to the nature of the path guidance constraints, which try to reach the goal by the shortest route possible). This is then compounded by the subsequent emphasis on length and erraticness optimisation. In the case of highly deceptive problems, the path needs to get longer before it can get shorter. Possibly the use of the proposed join operator which produces paths which always reach the goal point and which are also long would be advantageous here. In such cases, the emphasis on paths to get longer is reduced and the exploratory nature of paths is increased so the chances of avoiding the deceptiveness are also increased. Another alternative is to penalise paths, which pass over obstacles by adding a factor proportional to their constraint *iii)* and *iv)* values to the path length and guidance constraints. This will effectively make the paths appear to be longer than they are, making it easier for long paths to exist in the population and as a result reducing the need for the paths to grow to avoid the deceptive obstacles.

Obstacle clearance is not a factor in these constraints, although collision-free paths are produced in cluttered environments some of the paths pass along the edge of obstacles. This is not a problem if obstacle clearance elements are factored in as part of the initial discretisation process of the environment.

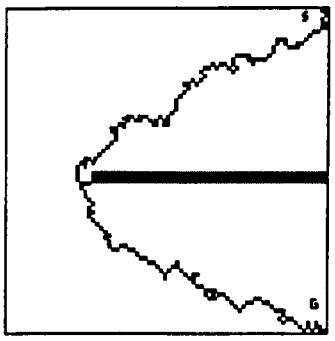


Figure 4.14 Sample path produced in a deceptive test environment.

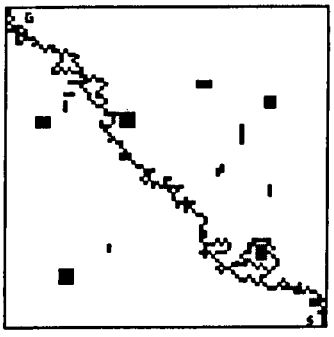


Figure 4.15 Sample path produced in cluttered environment.

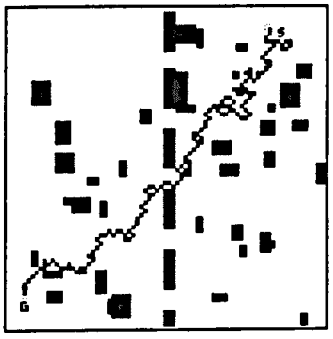


Figure 4.16 Sample path produced in a very cluttered environment.

### 4.3.2 Experiment II

*Aim:* To determine the essential characteristics of the path representation and operator set up in order to make path planning as easy as possible for the GA.

This section is concerned with establishing what properties are required to make the task of path planning easier for the GA. This requires an investigation of path representation and scaling methodology. With respect to the path representation, two new path structures are proposed and their performance compared with the default one as used in section 4.3.1. All the path representations are based on the relative path concept but each has its own distinctive attributes. Here the default path representation is referred to as *Fixed* and the two new representations as *nFixed* and *nDynamic* respectively. The structural properties offered by each of these path representations are summarised below along with their interpretation of the following movement sequence *N,N,E,SE,E,E,E*:

***Fixed*:**

Movements are in single units. The characteristics of this representation are:-

- Very flexible paths,
- Maximum path length is the same as the maximum number of direction changes,
- High maximum direction change density,
- Fixed length path for a given number of genes.

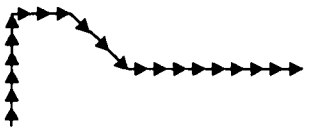
An example path using this representation [N,N,E,SE,E,E,E] is shown pictorially:

***nFixed:***

Can move in movements of n units. The characteristics of this representation are:-

- Fairly flexible paths,
- Long paths possible, for a given chromosome length,
- Maximum path length exceeds maximum number of direction changes,
- Low maximum direction change density,
- Fixed length path for a given number of genes.

An example path using this representation [N,N,E,SE,E,E,E] is shown pictorially, where n=3:

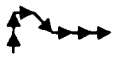


***nDynamic:***

The number of movements in any direction are dynamic in the range of 1..n. The characteristics of this representation are:

- Flexible paths,
- Long paths possible, for a given chromosome length,
- Low maximum direction change density,
- Variable length path for a given number of genes,
- Compact path representation.

An example path using this representation [(2,N),(1,E),(1,SE),(3,E)] is shown pictorially, where n=4:



Once the appropriate path representation has been determined, the test will switch to establishing the most appropriate scaling operator. The scaling operator is responsible for controlling the rate of convergence of the population. Two possible choices are available linear and ranking. Linear scaling alters the fitness values of the population proportional to the best, worst and the average. However, ranked scaling replaces the fitness value by a rank index indicating the position of individual chromosomes in an order sequence based on their original fitness. In addition, the introduction of a post processing filter operator is investigated and the dependency on it for performance noted. This operator will remove any loops or backtracking from the final best path produced.

The obstacle filled test scenarios used in section 4.3.1 are used for the experiments, six test cases are constructed, two in cluttered environments and four in deceptive environments, the parameters for which can be seen in the table 4.10. Each scenario is evaluated 16 times using a different set of initial random seed values for the GA operators. A HP-700 series workstation is used to run the tests.

| TEST ID | START LOCATION | END LOCATION | SCENARIO | OPTIMAL PATH LENGTH | OPTIMAL PATH SMOOTHNESS |
|---|---|---|---|---|---|
| 1 | (79,0) | (79,79) | 1 | 121 | 7 |
| 2 | (0,0) | (79,79) | 1 | 107 | 3 |
| 3 | (79,79) | (0,0) | 2 | 91 | 3 |
| 4 | (70,38) | (70,44) | 1 | 104 | 5 |
| 5 | (30,20) | (75,64) | 1 | 80 | 4 |
| 6 | (65,5) | (4,70) | 3 | 70 | 4 |

Table 4.10 List of test attributes for main suite of cluttered environment test cases.

#### 4.3.2.1 Results from experiment II

Path planning failures occurred in highly cluttered and highly deceptive test cases (tests 6 and 4 respectively). Those in the cluttered environment resulted from the inability to move the path off the corner of one or more obstacles and those in the deceptive case resulted from the sheer deceptiveness of the problem. A typical failure in this latter case can be seen in figure 4.23.

For the *Fixed* representation, the average time taken to find the first collision-free path was 85 generations. As before these initial paths were long and haphazard. The failure rate in the highly deceptive case (test 4) was 100% and in the highly cluttered was around 6%. The final best paths suffered from the same problems as those reported in section 4.3.1.

For the *nFixed* case it was found that there existed only a few valid values for **n**. The representation reduces the resolution of the environment depending on the value of **n**. However some values of n would create areas within the environment which could not be reached by any point in the path. This is not a problem if one wants to constrain the usage

of the planner (it would indeed lead to a more efficient planner) but in this case, we want it to be applicable to all environments and all cases. Assuming a square environment the validity of an environment's resolution of values for **n** are determined by the following inequality 0< A >= B, where **A** is the number of accessible points in an enclosed area and **B** is the area of the environment and are defined as follows:

A=(6*n)-1.

B=(n+1)2.

Only valid values for **n** were considered. Comparing the paths produced under these circumstances and with those of the *Fixed* representation it was found that the larger the value of **n** the more efficient the final paths produced were. However the larger this value the greater the number of failures in the highly cluttered test scenarios. The average time taken to find initial collision-free paths was 41 generations. The failure rate in the highly deceptive test case was about 92% and for the highly cluttered case 13%. A value for **n** of 3 was found to give the most consistently good results. The initial paths found were a lot less haphazard but generally longer. The final optimised paths were a lot less erratic and typically shorter. The direction change density (DCD, a measure of the ratio of path length to the number of direction changes in it) was again better with an improvement of about 70%. The optimisation duration for paths also increased to around 165 generations. Although the duration was longer, the number of improvements produced fell. The computational performance also improved to around 82 seconds. This was due to two factors: firstly finding initial paths earlier hence narrowing the search and secondly reducing chromosome length of paths in general.

For the *nDynamic* case there was no restriction on the value of **n**. However the larger the value the greater the computational demand processing took. A value of 8 was found to be adequate for the characteristics of the representation to be seen. Again, the performance of GA-planner using this representation is compared to the *Fixed* cases. The average time taken to find the first collision-free path dropped to around 31 generations. These initial

paths were a lot less erratic but longer. The final paths were always shorter and less erratic. The DCD for the final paths showed an improvement of around 70.9%. The failure rate in the highly deceptive case was around 81% and for the highly cluttered case about 13%. The optimisation duration showed a slight increase to around 140 generations and the improvements in paths were both abundant and regular. With the aforementioned value for **n**, the computational performance fell to about 85 seconds for the same reasons as in the *nFixed* case.

A comparison of performance between the *nFixed* based GA-planner and the *nDynamic* based one showed that the overall performance of the two was always quite close. However, the *nDynamic* always had the edge particularly in reducing the failures in deceptive cases and with respect to consistency.

The comparison of performance using the scaling operators indicated that there was not one outstanding scaling operator and that performance was representation dependent. The improvement in performance was small but the most noticeable area of benefit was the reduction in the number of failed runs in the deceptive cases. Performance was best for *Fixed* representation when ranked scaling was used; for *nFixed* representation when linear scaling was used and for *nDynamic* representation when ranked scaling was used.

The final paths when post filtered in all cases of representation showed some improvement. However, the dependency on this operator for performance for planners based on the *nDynamic* representation was lowest and highest in the case of the *nFixed* representation based planners.

Some sample paths produced in these tests using each representation can been seen in Figures 4.17 to 4.23. These give a visual indication of how the representation used alters the characteristics of the paths produced. Tables 4.11 to 4.14 show the average performance of the GA-planner based on each of the three representations, in each of the test cases. Table 4.11 shows the time taken to find the first collision-free path in each case. Tables

4.12 and 4.13 show the excess: path length and erraticness respectively of the best path found. Table 4.14 shows the DCD values for the best path in each of the tests.

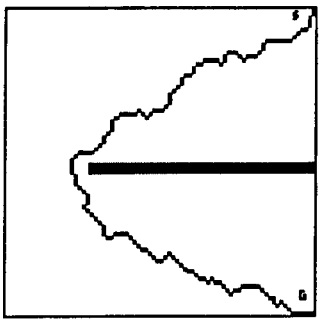


Figure 4.17 Path produced using Fixed based planner in test 1.

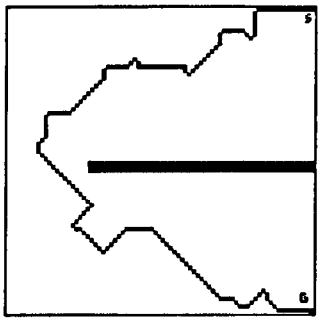


Figure 4.18 Path produced using nFixed based planner in test 1.

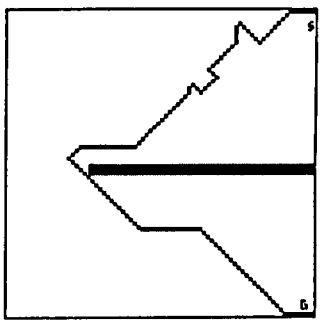


Figure 4.19 Path produced using nDynamic based planner in test 1.

Figure 4.20 Path produced using Fixed based planner in test 6.

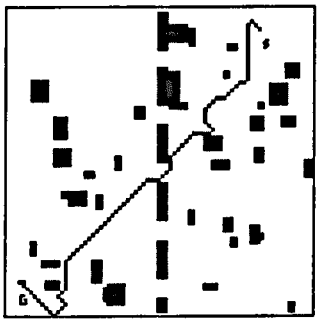


Figure 4.21 Path produced using nFixed based planner in test 6.

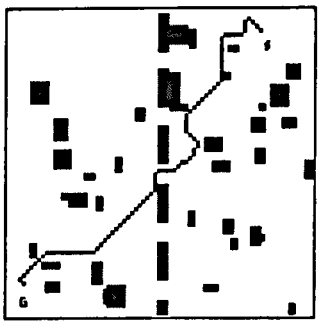


Figure 4.22 Path produced using nDynamic based planner in test 6.

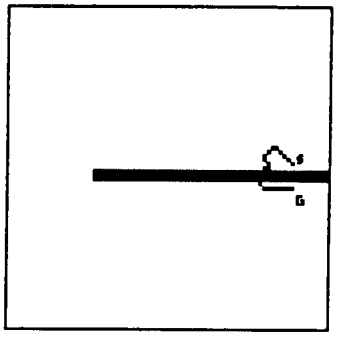



Figure 4.23 Typical failing path in test 4.

| | | TESTS | | | | | |
|---|---|---|---|---|---|---|---|
| | | **1** | **2** | **3** | **4** | **5** | **6** |
| Fixed | Mean time taken to find first collision-free path (in generations). | 110 | 52 | 37 | | 113 | 115 |
| | Standard deviation. | 19.87 | 15.583 | 5.438 | | 56.216 | 54.938 |
| nFixed | Mean time taken to find first collision-free path (in generations). | 38 | 14 | 19 | | 4 | 130 |
| | Standard deviation. | 16.402 | 5.085 | 8.731 | | 2.913 | 58.901 |
| nDynamic | Mean time taken to find first collision-free path (in generations). | 15 | 12 | 9 | | 5 | 114 |
| | Standard deviation. | 5.418 | 2.848 | 5.211 | | 1.871 | 28.31 |

Table 4.11 Mean time taken for GA-planners based on various path representations to find first collision-free path in obstacle filled test environments and their associated standard deviations.

| | | TESTS | | | | | |
|---|---|---|---|---|---|---|---|
| | | **1** | **2** | **3** | **4** | **5** | **6** |
| Fixed | Mean excess path length. | 100 | 63 | 60 | | 72 | 85 |
| | Standard deviation. | 34.415 | 54.954 | 24.604 | | 83.192 | 55.330 |
| nFixed | Mean excess path length. | 120 | 58 | 61 | | 78 | 63 |
| | Standard deviation. | 73.286 | 54.715 | 51.236 | | 41.161 | 52.945 |
| nDynamic | Mean excess path length. | 90 | 43 | 45 | | 44 | 46 |
| | Standard deviation. | 53.925 | 71.337 | 76.722 | | 36.655 | 53.996 |

Table 4.12 Mean amount of excessive path length in the best path found by GA-planners based on various path representations in obstacle filled test environments and their associated standard deviations.

| | | TESTS | | | | | |
|---|---|---|---|---|---|---|---|
| | | **1** | **2** | **3** | **4** | **5** | **6** |
| Fixed | Mean number of excess direction changes in the best path. | 130 | 96 | 96 | | 86 | 120 |
| | Standard deviation. | 31.704 | 49.284 | 26.125 | | 72.896 | 48.097 |
| nFixed | Mean number of excess direction changes in the best path. | 79 | 42 | 39 | | 40 | 60 |
| | Standard deviation. | 22.885 | 16.449 | 16.797 | | 11.526 | 25.014 |
| nDynamic | Mean number of excess direction changes in the best path. | 50 | 42 | 35 | | 20 | 33 |
| | Standard deviation. | 18.210 | 27.787 | 22.592 | | 14.269 | 15.249 |

Table 4.13 Mean amount of excessive direction changes in the best path found by GA-planners based on various path representations in obstacle filled test environments and their associated standard deviations.

| | | TESTS | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 |
| Fixed | Mean direction change density in the best path. | 0.71 | 0.69 | 0.7 | | 0.74 | 0.81 |
| | Standard deviation. | 0.0386 | 0.0512 | 0.06 | | 0.05 | 0.0363 |
| nFixed | Mean direction change density in the best path. | 0.25 | 0.2 | 0.24 | | 0.25 | 0.28 |
| | Standard deviation. | 0.0243 | 0.0344 | 0.0261 | | 0.0254 | 0.0291 |
| nDynamic | Mean direction change density in the best path. | 0.23 | 0.201 | 0.19 | | 0.15 | 0.18 |
| | Standard deviation. | 0.0301 | 0.0394 | 0.0421 | | 0.0532 | 0.0301 |

Table 4.14 Mean direction change density in the best path found by GA-planners based on various path representations in obstacle filled test environments and their associated standard deviations.

#### 4.3.2.2 Discussion of results from experiment II

The experiments carried out have shown that as with most GA based problems, for path planning the way the path is represented has an impact on its solvability. The results identified three characteristics which are of importance in a path representation. Firstly the *step size*, which is the maximum permissible distance a path segment can travel before a change of direction can occur. This is related to the representation's ability to produce smooth paths, the larger the step size the smoother the path. Secondly, the *direction change flexibility*, which is the number of direction changes that can occur within a given distance. This contributes to a representations ability to shorten path lengths, the more flexible, the smoother the path. Thirdly the *step size flexibility*, which is the magnitude by which the distance direction change can vary. This, too, affects the representations ability to shorten

paths, where greater flexibility leads to short paths. The nDynamic representation possessed the majority of these hence its good performance.

The use of scaling offered its main benefits in reducing the number of failures in deceptive cases. Those representations, which benefited from the use of the ranked scaling in these cases required slower convergence times. By ranking the chromosomes in this form, initial paths which get to the goal and fall pray to deceptive traps in the environment always have a roughly constant difference in fitness from those which avoid the traps but don't reach the goal. This gives the latter paths more time to get to the goal. It also allows more time for variants of those paths which reach the goal to be produced which avoid more of the traps, which suggests that perhaps that flexible paths require more time to avoid deceptive obstacles. Those, which benefit from the use of linear scaling, require just that the best path be constrained by the average path.

The post filtering of the best paths produced more tidy paths. Although all representations benefited from its application, the *nFixed*-based planner was the most dependent on it for its performance level. By using fixed step sizes, it predisposed the representation to making long paths and possibly ones with excessive loops in them. Since in order to get to particular point in its $n \times n$ resolution it often has to cover this whole area completely with motions to ensure it passes through the point. Another is that it may need to move away from the goal and loop back if it is not a multiple of n away from the start point. The *nDynamic* based planner was least dependent as there was very little need to move away from a point in order to reach it and also because the length and shape of the path could be altered dramatically by a few changes to the magnitude portion of a genes field. This makes the paths easier to optimise. Again, flexibility in path step size is a benefit when it comes to dependency on filter operator for performance.

### 4.3.3 Experiment III

*Aim:* To determine the most appropriate method for improving the robustness of the GA based planner.

Following the results of the experiments in section 4.3.2 the default planner was re-configured to use ranked scaling, post path filtering and to be based on the nDynamic representation, with **n** set to 8. In this section, the experiments were focused on improving the general robustness of the planner. Three routes were available which could lead to improvements in robustness, these being:

1. constraint reformalisation,
2. intelligent operators,
3. additional initial pre-processing of environment.

Each of these three alternate routes was investigated. The first route, re-formulation of the planners constraints, would take the form of inspecting what qualities the existing constraints explicitly or implicitly encourage during path planning and comparing these with those constraints which are believed to be either explicitly or implicitly required for the task. The explicitly stated constraints either can be over or under defined for the task or their respective functions may be incorrectly defined for certain path states. As with most implicit constraints that exist (in the present set up) they are not precisely governable, one cannot predict when or how they are processed. So some benefit may be drawn from making them more explicit.

Implicit relationships exist between several of the current constraints, these are *ii), iii), iv), vi)* and *vii)*. The relationships are:

- Distance from the goal and collisions. If a collision occurs, then the path has to travel over the obstacle to some degree. The resultant distance-to-goal calculation indicates that the path is closer to the goal than it actually is.
- Path length and collisions. Similarly, if the path passed over an obstacle a corresponding increase in its length will occur compared with no obstacle.
- Path length and smoothness. As a path gets shorter its maximum erraticness decreases. Also for a path to reach its optimal length implies it has reached its optimal smoothness.

All of these implicit constraints except the last one can benefit from being explicitly stated. The relationship in the latter is already to some degree controllable since the relationship here is a physical one. This explicitness is accomplished by introducing a factor associated with collisions into constraints *ii)*, *iii)* and *vi)*. The factor selected was the collision avoidance sum, constraint *v)*, owing to its discriminative abilities, which offer differing penalties for different obstacle traversals, a proposal made in section 4.3.1.

Further, with regard to the explicitly stated constraints, a re-evaluation revealed that the existing constraints were understated. This was a consequence of the under specification of the original definition on which they were based, which was to produce optimal collision-free paths when it should have been to produce safe optimal collision-free paths. In order for safe paths to be generated, the path needs to keep some clearance distance between the path and the obstacle. To achieve this an additional constraint is required. Its priority needs to be higher than constraints *vi)* and *vii)* in order to be effective, a proposal made in section 4.3.1.

The second route to be investigated concerns the use of intelligent or problem-specific operators in the GA-planner. Two operators are proposed, the first is an extension of the previously mentioned filter operator (section 4.3.2) allowing it to be used on the populations of paths throughout the run so long as the path reaches the goal, and the second is a join operator (proposed in section 4.3.1) which generates initial paths based on the

random variations of the optimal path and which are made to move away from the start point initially.

The third and final route to be investigated, additional initial pre-processing of the environment, introduces a new obstacle avoidance scheme which works in conjunction with the existing directed flow avoidance method. This new scheme uses the concept of avoidance contours and requires additional information about the number, position and shape of the obstacles in the environment before any processing can begin. This method was initially derived from the GA-planner's inability in some cases to remove paths from the edges of obstacles. The generalisation of it so as to handle any collision circumstance is investigated here. Owing to memory restrictions, only a finite number of obstacles can be avoided at any time using this method (maximum of 2) necessitating the need for the existing method to remain. The scheme relies on the defining of, for each obstacle, a collision-free path which encompasses the obstacle. This path is called a contour and it belongs to the obstacle. The shape of the contours employed may be governed by one or more of the following factors:

- Whether smooth trajectories around obstacles are desired.
- Whether smooth paths are required around obstacles.
- If mandatory obstacle clearance distances are used.
- Whether the minimising of the chances of violating the contours conditions (as stated below) is to be ruthlessly enforced.
- Whether a constant scaleable shape is desired for all contours.

These factors may be globally applicable to all obstacles or may vary amongst them. The application of the contour information generated from pre-processing gives rise to the avoidance scheme. An obstacles contour is only considered when a collision occurs on it. For the contour to be used, two conditions must hold true:

i) The path must have entry and exit points onto the obstacle.

ii) The path must have entry and exit points onto the contour.

If any of these conditions is not true then the avoidance scheme fails. The first condition will only be violated at the end of a path. This implies that the representation does not allow for long enough paths and should be altered. The second condition can be violated if the obstacle's contour leaves space between it and the boundary. This violation will also only occur at the end of a path. The remedy is to ensure the path representation's maximum length is appropriately defined. If the two conditions hold then the contour can then be used to generate an avoidance path around the collision on the obstacle. This path is formed by finding a path between the entry and exit points on the contour. Such a path will always exist if no part of the obstacle is touching the boundary. If the obstacle does touch the boundary then the contour formed for it will be discontinuous. If no path can be generated then the area is inaccessible. The generated avoidance path is used to replace the section of path between the contours entry and exit points which causes the collision and passes over the obstacle.

As mentioned previously, there are several possible factors which may influence the shape of the contours used. However, in this work the only factor considered is the reduction of the possibility of violating the contour conditions. This results in contours whose shapes follow and touch the boundary of their respective obstacle. The reasons for this are:

- The first condition mentioned above will hold true if a point in the path reaches the goal.
- The second condition will be true in the same circumstances just as long as an obstacles avoidance contour follows and touches its boundary.

The routes undertaken to improve the robustness of the GA-planner all have a commonality in their approach to this end, namely the encouragement, production or sustainability of long paths. This avenue of improvement will seem to herald a deterioration

in computational performance. To counter or minimise the effect of this, an existing passive constraint was promoted in all GA configurations used in this section. The passiveness of the constraint is maintained but it has an activation trigger associated with it to promote and demote its role. The constraint is the fixed length nature of the path representation. By appropriately reducing this value, the constraint becomes dynamically active. Its constraining effect reduces the amount of path which is permissible to find optimal paths hence reducing the maximum processing time for wayward paths. The constraint only becomes active once a collision-free path has been found. The longest collision-free path in each generation then defines the longest allowable path.

Each of these stages was tested using the same test suite used in section 4.3.2 and each test was run 16 times, using different initial random seeds on each run. The experiments were all carried out on a HP-700 series workstation.

#### 4.3.3.1 Results from experiment III

The re-formulation of constraints reduced the failure rate of the GA-planner to around 50% in highly deceptive cases and 0% in all others. There was a predisposition for generating longer paths but this led to finding the first collision path slightly earlier (average of 3 generations) and more consistently than the non-reformulated GA-planner. The final filtered paths were slightly longer and more erratic as well. Also, due to the need for obstacle clearance, paths tended not to weave between obstacles owing to a preference to circumvent them, leading also to longer but less erratic paths. This also limited the number of distinct routes to the goal. The need to calculate clearance values also hampered computational performance although it did show a marked improvement with the average run time being around 75 seconds.

The use of intelligent operators resulted in a reduction in the failure rate of the planner. However, the level of this reduction varied, dependent on operator combination, although the use of the filter operator consistently led to the best results. As a whole, the failure rate

in both the highly deceptive and cluttered cases had a maximum value of 13%. The computational performance also varied dependent on operator combination, again with the use of the filter operator resulting in the best performance. For the Join operator alone this was an average 63 seconds, for the filter operator alone this was an average of 54 seconds and for the combination of the two it was an average of 58 seconds. The use of the join operator reduced the time taken to find the first initial collision-free path, however these paths were the longest and most erratic. The initial paths are best when the filter operator is used and the final best paths were also on average found earlier. The best final paths were produced when both the join and the filter operator were used in conjunction with each other. The general level of erraticness was a lot less when these operators were used.

The use of the additional obstacle avoidance method resulted in a total nullifying of the robustness issue. Here, no test failed and initial collision-free paths were always found earlier, on average around generation 11. These initial paths were less erratic but had very poor obstacle clearance distances, as a result of the avoidance method. They were also generally longer. A longer optimisation duration was created together with a large number of valid paths, however optimisation of paths was slower. The real-time performance improved to give an average runtime of 67 seconds.

The use of the passive constraint was one of the key factors responsible for increased computational performance, although in some cases it did limit the exploration of alternate routes (the re-formulated constraint case) in the environment because a path could not expand and contract it offered great advantage to those approaches which generated initial paths quickly, smooth paths, short paths, or a lot of collision-free paths.

A small number of paths produced in this section can be seen in figures 4.24 to 4.26. Tables 4.15 to 4.17 show the average performance produced by the GA-planners based on each of the alternative routes for improvement, in each of the six test cases.

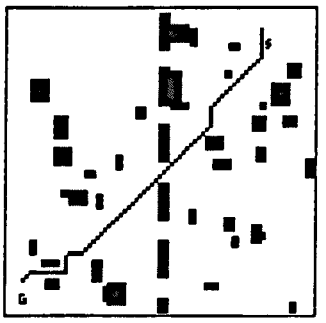


Figure 4.24 Path produced using contour avoidance system in test 6.

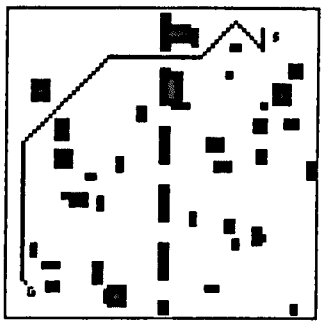


Figure 4.25 Path produced using constraint reformalisation in test 6.

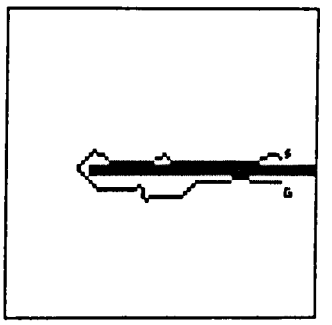


Figure 4.26 Path produced using contour avoidance system in test 4.

| | | TESTS | | | | | |
|---|---|---|---|---|---|---|---|
| | | **1** | **2** | **3** | **4** | **5** | **6** |
| Route 1 | Mean time taken to find first collision-free path (in generations). | 21 | 12 | 15 | 145 | 5 | 91 |
| | Standard deviation. | 12.189 | 4.630 | 3.04 | 18.866 | 3.13 | 33.1 |
| Route 2 | Mean time taken to find first collision-free path (in generations). | 14 | 3 | 18 | 38 | 7 | 79 |
| | Standard deviation. | 5.129 | 1.372 | 4.25 | 12.891 | 2.197 | 13.048 |
| Route 3 | Mean time taken to find first collision-free path (in generations). | 18 | 10 | 12 | 14 | 3 | 45 |
| | Standard deviation. | 4.758 | 2.371 | 3.157 | 3.981 | 1.213 | 8.971 |

Table 4.15 Mean time taken to find the first collision-free path using various GA-planner improvement routes in obstacle filled test environments and their associated standard deviations.

| | | TESTS | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 |
| Route 1 | Mean excess path length. | 80 | 44 | 40 | 161 | 49 | 54 |
| | Standard deviation. | 18.114 | 18.036 | 15.75 | 30.187 | 16.466 | 22.119 |
| Route 2 | Mean excess path length. | 49 | 27 | 15 | 158 | 44 | 24 |
| | Standard deviation. | 9.721 | 8.759 | 3.938 | 31.445 | 6.687 | 7.002 |
| Route 3 | Mean excess path length. | 65 | 31 | 26 | 45 | 51 | 22 |
| | Standard deviation. | 8.532 | 9.313 | 9.501 | 18.84 | 7.915 | 9.629 |

Table 4.16 Mean amount of excessive path length in the best path produced using various GA-planner improvement routes in obstacle filled test environments and their associated standard deviations.

| | | TESTS | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 |
| Route 1 | Mean direction change density in the best path. | 0.22 | 0.19 | 0.18 | 0.19 | 0.19 | 0.19 |
| | Standard deviation. | 0.029 | 0.053 | 0.047 | 0.067 | 0.049 | 0.031 |
| Route 2 | Mean direction change density in the best path. | 0.1 | 0.085 | 0.051 | 0.13 | 0.11 | 0.12 |
| | Standard deviation. | 0.0156 | 0.01 | 0.008 | 0.023 | 0.038 | 0.042 |
| Route 3 | Mean direction change density in the best path. | 0.23 | 0.163 | 0.17 | 0.01 | 0.19 | 0.191 |
| | Standard deviation. | 0.0291 | 0.048 | 0.069 | 0.038 | 0.048 | 0.037 |

Table 4.17 Mean direction change density in the best path produced using various GA-planner improvement routes in obstacle filled test environments and their associated standard deviations.

#### 4.3.3.2 Discussion of results from experiment III

Each of the three routes proposed contributed in varying amounts towards the improved robustness of the GA-based path planner. As mentioned earlier, there exists a commonality between the approaches and hence it is possible to conclude that it is this factor which appears to be the main aspect with regard to robustness. However, this commonality of long path generation is contradictory to the aim of the planner, so as well as having this ability, it must also have some method of controlling its use. This will aid in the optimising of paths. The control system of route 1 was dependent on collisions, such that if there were no collision then there would be no additional encouragement for generating long paths. The control for route 2 was dependent on the operator. For the join operator it was in the initialisation process only, these control factors were governed by parameter set up. With respect to the filter operator control, it was governed by: whether the path reached the goal and whether the path contained an loops or backtracks. In these cases, the path would start to get shorter. In the case of route 3, control was triggered by obstacle collisions, if no collisions occur then paths got shorter, the emphasis here being on more efficient obstacle avoidance. All these control systems were aided by the use of the passive constraint, which encouraged the exploration of shorter less erratic paths, its level of contribution is obviously dependent on the stringent nature of the control system. Of each approach, route 3, (the introduction of an additional obstacle avoidance method) was the only approach to eliminate the robustness issue altogether. The performance here was however not the best. Route 2 (the use of intelligent operators) offered the best performance but was less robust. A combination of routes 1 and 2 still did not deliver the level of robustness offered by route 3. The combination of all 3 routes however seems to offer the most benefit, i.e. the robustness of route 3 together with the performance of route 2. It also offers multiple ways in which to tackle robustness hence making the system even more robust as well as increasing diversity in solutions to deceptive or complicated test cases.

With the efficiency of the contour-based obstacle avoidance method, it could be argued that the GA is effectively just navigating in free spaces. There is some truth to this,

however, existing path planning techniques (such as graph or network based planners) do a similar thing whereby obstacles are effectively removed from the environment. However, owing the restriction on the number of obstacles that can be avoided at any one time, the GA still has a degree of obstacle avoidance responsibility.

The use of the join operator makes it easier to find initial paths. However, it does not aid in their optimisation. The filter operator however does aid in their optimisation but it doesn't encourage long paths. What it does do is reduce the likelihood of collision by removing any unnecessary points in the path. This removal of unnecessary points in the path also leads to populations of shorter paths which in turn leads to better computational performance.

As a side effect of addressing the robustness issue there has been an associated enhancement in performance (speed and path efficiency). This improvement can be attributed mostly to utilisation of the passive constraint of the system. This method resulted in a steadily reducing path length since the GA was directed to target its search for optimal paths within a decreasing range. The computational performance gain is achieved by the reduction in chromosome processing generated by the reduction in path length. The activation of this constraint also has a positive effect on the optimising performance of the planner. Also, the amount of time saved by not spending prolonged periods of time processing solutions which will never reach the goal also added to the performance level.

### 4.3.4 Experiment IV

*Aim:* Evaluation optimum configured GA-planner.

Having established the requirements for optimal performance and speed as well as robustness of the GA-planner, its relative standing as a viable alternative to existing path planning approaches had to be established. Two forms of the GA-planner were used, which differed only in the inclusion of the contour avoidance method. The argument that the use of the contour avoidance method made the GA-planner a free-space navigator was the

reason behind this distinction. By comparing the level of performance both with and without the avoidance method, it is possible to determine the most competitive role for the GA, as a free-space navigator or as a full-blown obstacle and free-space navigator. To determine its competitiveness it was evaluated against three existing methods using test scenarios, which were compatible with all the methods. The three approaches evaluated were:

- Potential field,
- Configuration space,
- Wave propagation.

The metrics considered were computational demand, efficiency of paths produced, availability of paths and predictability of paths generated. Each method was run twice (with the GA-planner using different initial random seeds on each run for its operators) and within each test the six test cases were used, and average performance statistic gathered. All the tests took place on a HP-700 series workstation.

#### 4.3.4.1 Results from experiment IV

The GA-planner using the first configuration (no contour avoidance method) did not offer predictable solutions across each of the two runs performed for each test. The best paths produced in each of the runs had different fitness values and slightly different routes through the environment. Multiple valid potential solutions were produced with fitness values worse than the best and whose paths took distinctly different routes through the environment. Considering only the members of the final population it was found that, an average of 11% of them were within a 10% range of the best paths cumulated fitness. Of these, an average 35% used similar routes through the environment as the best path. The paths exhibited good obstacle clearance and path smoothness, however as a result their lengths were not near the optimal. Problems with highly deceptive test cases were evident from increased search time to find the first path. However, no failures occurred. It took on

average 39 generations to find the first path and the average run time of the planner was 66 seconds.

The GA-planner using the second configuration (contour avoidance method) again did not offer predictable solutions in any of the runs of each test. However a greater number of valid paths were found and the number of distinct routes was greatly reduced. In the final population, an average of 37% of the members were within 10% of the best paths summed fitness. Of these, an average of 43% of the paths used similar routes through the environment as the best path. The obstacle clearance was not as pronounced as in the first configuration, as a result of which their fitness value was slightly worse but the length of the paths was closer to the optimal. The paths were smooth and the initial paths were found on average around generation 9. The average run time was 47 seconds, and the optimisation period 160 generations, with, on average, the last 30 generations of optimisation producing very little change in the fitness of the best path. Deceptive cases presented no problem, performance was fastest in sparse environments, and the best paths were generated in cluttered environments.

In order for the potential field method used for comparison to plan paths which avoided the local minima around closely placed obstacles in the cluttered test cases, either random background noise had to be introduced to the system or a reduction in the extent of the repulsive force of obstacles had to be implemented. However, neither of these approaches were sufficient to provide a way of escaping from the local minima that were encountered in the deceptive test cases. As a result, the performance of this approach was very poor in even the easiest of the deceptive test cases. This poor performance is due to the inherent local nature of the planning strategy of the potential field approach, which is stark contrast to the global overview that is required to re-direct paths in order to achieve success in deceptive cases. However, in those instances when successful paths were produced it took an average of 1.45 seconds to generate the path, and in each test case only a single path was produced and that path was reproduced on any subsequent run of the test.

The wave propagation method consistently produced valid paths in all the test cases, also, these paths were also always very close to optimal in length. Owing to their optimal length their obstacle clearance was very poor. For each of the two runs of each test, the same path between start and goal was consistently produced for each run. Also these paths were always generated in the same amount of time and no additional paths were ever produced. This method was very quick, taking an average of 0.35 seconds to generate a path. The speed of processing was directly proportional to the number of obstacles in the environment (i.e. the greater the amount of occupied space in the environment the faster the processing). The best computational performance was realised in cluttered environments and the best path planning performance was realised in sparse environments. The deceptive test presented no problem to the method.

The configuration space method required a dual stage planning process. First the environment was transformed into its configuration representation and secondly this representation was searched for paths. The transformation stage took around 61.47 seconds. The searching stage took on average 75 seconds. Given an overall path planning time of 136.47 seconds, only a single path was produced in each test and this was returned consistently throughout the runs of a test.

From these results, it can be seen that the potential field and the wave propagation techniques are less computationally demanding than the GA-based planners and that the configuration space is most demanding. This comparison was based on an arbitrary generation setting for the GA-based planners. The results from runs in which elapsed time is used as the stopping criteria, and not elapsed generations offer only a single mode of comparison, the effectiveness of paths produced in the time frame. The duration here was set to 3 seconds, which would bring the computational time within the same time frame as the slower of these two main methods. The performance of the first GA-planner was poor under these circumstances. It very rarely managed to find a collision-free path in the time and never in highly deceptive cases. Those paths, however, which were found were very poor, (long and highly erratic). However the second configuration (using the contour

avoidance method) performed a lot better, although it did not always find a path (only in highly cluttered cases). The paths produced, typically had undergone about 2 to 4 generations worth of optimising before the time limit expired. The quality of the paths, though, was much worse than those produced by the faster methods.

#### 4.3.4.2 Discussion of results from experiment IV

The performance of the GA-planner is best when the contour avoidance method is used. Also the best paths produced by the planner are comparable to those produced by the other methods. Further, the GA-planner is the only method that produces multiple potential paths for a given problem. However, this is done in a non-deterministic manner. Some of these potential paths, use distinct routes from the best path. This ability is not a problem if what is needed from a planner is a degree of flexibility in the sense that, if a real world obstacle does not correspond to its exact location in the environment map used to generate the path, the system will not fail, or if an area the path passes through is blocked for some reason. By providing slight variations of a path the GA-planner makes it possible to handle cases where the position of obstacles varies slightly and by providing paths which take alternate routes, make it possible to handle blocked route cases. If this flexibility is not there, then under such circumstances the planning process will have to begin a fresh using the new environment layout. However if what is required is a deterministic system in which no room for ambiguity can exist, then this is not a good quality. The other main difference is in time, where the GA-planner lagged behind the potential field and wave propagation methods. Also when using time as the stopping criteria of the GA-planner the paths produced were nowhere near optimal, in most cases they were in the later processes of finding the first path or just beginning the optimisation process on the first path. However, although the potential field methods offer better computational performance this is only on a restricted set of test cases. That is, as long as the path planning case contains no deception in it this benefit will be realised. In this respect the GA-planner offers better robustness than potential field approaches in these cases although its overall computational performance is worse. In path planners robustness is always preferred to speed. Under the time restricted

evolutionary test runs, the performance was again best when the contour avoidance method was used, which was due to the rapid amount of optimisation that takes place in the early to mid stages of the path optimisation process. The performance of this configuration could be speeded up by reducing its population size. However this could lead to a potential loss of diversity in the population (leading possibly to premature convergence), a limitation on the rate of search space traversal (leading possibly to increased run time) and a reduction in alternate routes (leading possibly to reduced flexibility). If the speed of the computer's processor is not a restriction, and flexibility, robustness and quality of path are the main assets required in a path planner, then implementing this configuration of the planner on a faster machine or a parallel architecture would produce a planner of comparable path planning ability as those investigated here. The results also show that the GA-planner is better when its obstacle avoidance burden is reduced, this can be seen in the improved performance offered by the contour avoidance configuration. This suggests that another alternative to speeding up path planning is to make the planner a wholly free-space based system.

## 4.4 SUMMARY

In summary, this research has developed an evolutionary method based on a genetic algorithm and shown it to be a viable tool in the planning of collision-free paths for mobile robots. The following points have been identified:

- The use of a relative path representation allows for collision-free paths to be planned and optimised in static, known and cluttered environments.

- Prioritised optimisation of multiple constraints can be handled with a vector fitness representation and a dominance based ordering system. This allows genetic material from colliding paths and other constraint violations greater opportunity to contribute to the next generation.

- The characteristics required for path representation in order for more optimal paths to be produced.

- The nature of the environments and path planning situations which present the greatest problem for the GA to solve. It has also shown that they can be addressed with varying degrees of success, using constraint re-formulation, intelligent operators, avoidance contours or some combination of the three.

- Robustness problems can be almost eliminated by the use of avoidance contours.

- Paths can be planned in environments containing arbitrarily shaped obstacles.

- Time and path optimisation improvements can be gained from converting the static passive fixed length nature of the path structure into a dynamically active constraint.

- The path planning performance of the GA based planner is comparable to that of existing techniques. However, its computational demand is a lot more than planners based on both potential field and wave propagation methods.

# Chapter 5

# INTERPRETER AND SIMULATED ENVIRONMENT

## 5.1 INTRODUCTION

The theme of this thesis now investigates the application of another evolutionary technique to the development of mobile robot systems. This work considers the role communication can play in evolving software control systems. This chapter introduces the tools developed for to undertake this work. The systems will be generated by an evolutionary technique known as Genetic Programming, The scenarios will be multi-robot.The parameter settings used in the configuration of the GP and simulated environment are introduced in section 5.2, the set-up of the GP-engine is covered section 5.3, while the remaining sections of this chapter are dedicated to presenting the robot and environment model.

## 5.2 Preliminary experiments

The application of genetic programming necessitates the selection of a number of parameters relating to both the process of evolution and the application itself. Those parameters which are particular to a certain task are covered in the appropriate chapters. However, those which are common to all the tasks are introduced here and are listed below with their associated values:

- Maximum generations. 300
- Population size. 130
- Number of crossover points. ProgramLength1+ProgramLength2
- Mutation rate. 0.05

- Perturbation rate. 0.5
- Resource allocation percentage. [70, 30]
- Initial depth of trees. 6
- Interpreter time-slice duration. 6
- Robot initial energy level. 50

The values of most of these parameters were determined by conducting a preparatory set of experiments aimed at configuring the GP into an effective and general set-up. A major concern was the resultant processing time a particular parameter setting or combination of parameter settings would produce. Each of the aforementioned parameters was influenced by different factors which are described below.

Preparatory experiments showed that in most cases the most significant level of optimisation had been completed by generations 280 to 295, so the termination criteria was set to 300 generations.

The compiler memory segment limit placed an upper bound on the size of the memory resident population. However, experiments showed that a level of 130 was effective in producing programs of a diverse nature, while minimising processing requirement.

The number of times crossover is performed in any given application of the operator is proportional to the average length of the two programs involved. This average program length proportional level was used to help ensure that the programs produced as a result of the crossover process were more or less of the same size. Initial experiments highlighted the fact that if a fairly small program undergoes crossover with a large one, the large program tends to consume the smaller one, leaving perhaps a terminal or a single operator behind. This on the one hand produced offspring too small to do anything meaningfull and on the other, offspring which performed well but were overly verbose. So by using the average length of programs to determine the number of crossovers increased the chance of smaller programs surviving and growing in size, which helped reduce the amount of wasted processing performed by the GP.

A high level of mutation, at 0.05, was required to encourage a better exploration of the function/terminal search space. This is because the GP's operators are based on swapping sub-trees (i.e. finding the best position for sub-trees). Very little effort is given however to the parallel requirement of ensuring they contain the optimal function configuration. This has a tendency of reducing the functional/terminal diversity of the populations. Initial experimentation found this mutation level to be effective.

A high perturbation rate of 0.5 was used to encourage a finer grade search as well as a more exhaustive search of the numeric constant terminal space of the problems. Again, this is due to the coarseness of the GP's genetic operators. Initial experimentation found this level of perturbation to be effective.

A resource allocation percentage set of [70, 30] was found in preparatory experiments to define broadly, an effective prioritising method when constraints were competing with each other as well as two a balanced reinforcement level when they were complementary.

The standard initial program depth of 6 was applied here (Koza [21]).

The time slice duration used by the interpreter was set as 6. This allowed, on average, a robot to perform 2 instructions before the interpreter swapped its program out. By doing so, the effective dynamics of the simulation is improved, a robot having just enough time to sense an appropriate change and begin to react to it before its time slice ends. Preparatory experiments found this level worked well.

In all of the experiments, each robot required an initial energy level, this level was set to 50 units. This energy level effectively defines the amount of time a test will last. This level of energy gave the individual robots sufficient time to cover the environment and perform actions appropriate to the task. Without unnecessarily increasing the run times.

## 5.3 GENETIC PROGRAMMING ENGINE

The preparatory experiments defined a GP with the following parameter set :-

- branch crossover allowing for variable number of crossover points (proportional to the average length of the parents,)
- mutation at a rate of 0.05,
- additional mutation through a perturbation operator for real numbers at a rate of 0.5,
- vector fitness,
- an initial program depth limit of 6,
- population size of 130,
- maximum of 300 generations of evolution,
- multiple code blocks (see section 5.5).

The reproduction method requires choosing 10 random members of the population and copying the best to the next generation ensuring the same individual is not copied twice. This process is repeated 16 times. The remainder of the population is formed by applying crossover and mutation to corresponding blocks within pairs of programs chosen on each constraint. All members of the new population including copied ones are evaluated on the current test scenarios.

The system incorporates vector-based constraints, which always have to be minimised. The bulk of the processing is always allocated to the first constraint (Table 5.1 shows the resource allocation used throughout the remaining work). In order to reduce the supervisory level an automatic thresholding system is utilised. The level of acceptable performance is stated in the form of a tolerance vector, which is hardwired in to the system and is unique for each task (the levels of which are given in the appropriate chapters). This tolerance vector causes all programs performing better than the prescribed level to be stored for later inspection.

| Constraints | i | ii |
|---|---|---|
| Resource allocation (%) | 70 | 30 |

Table 5.1 Percentage of population allocated to each constraint for reproduction in communication experiments.

## 5.4 ROBOT SET-UP

The simulation developed uses asynchronous continuously moving robots. The robots can turn up to 180 degrees in either direction and have a fixed turning radius. The robots are equipped with an attached light source and a set of light sensors which are able to categorise light detected into one of three sections within the viewing angle (these being left, centre and right). To achieve this each of the robots within the system is modelled independently, using the following information:

- radius (Rr),
- centre point (P),
- turning radius (Tr),
- visual distance,
- angle of view ($\theta$),
- light sensors (Ls, Cs, Rs),
- collision flag,
- current heading,
- destination heading,
- energy counter,
- a process record, which indicates position in program and other contextual information,
- random number stream,
- speed indicator.

The robots are modelled by a circle as shown in Figure 5.1, with a touch sensor at both the front and back, enabling them to detect obstacles. This information can be used to determine if a collision has taken place. The turning radius indicates how long as well

as how far a robot needs to travel before it can complete a 360-degree turn. The larger this value, the longer it takes.

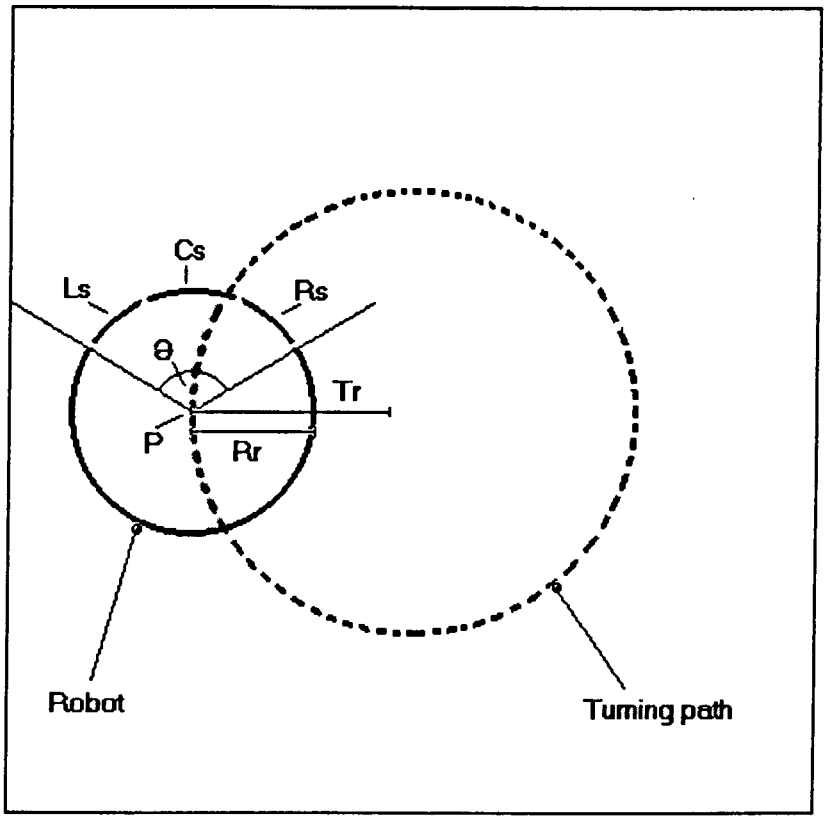



Figure 5.1 Modelling of robot and associated properties.

The robots are instructed to move by the following parameters, distance to travel, speed and turning angle. The turning angle is added to the destination heading. A robot moves in a straight line so long as its current heading is equal to its destination heading, otherwise it will follow a circular path. Its motion can be decomposed into straight line, angular motion or a combination of the two. If a negative distance is specified, it is interpreted as a straight-line reversal distance. Robots can use variable movement step sizes (movement distances), but an upper limit is placed on this. This maximum movement distance limit is required in order to keep performance consistent across various group sizes (for simulation purposes this is made proportional to the number of robots) as well as for realism. All angles are specified in radians.

Each robot has its own copy of the common program and a set of associated stack registers and system variables which are used by the interpreter when it is executing the robots associated program.

## 5.5 SIMULATOR AND ROBOT ENVIRONMENT MODEL

The evolution of all programs takes place within a single computer but their evaluation maybe distributed over several machines. Although the language for the evolved programs is unique, all other aspects of coding are implemented in C/C++. The complete system is run on a HP workstation running UNIX. Up to six additional machines are involved in the distributed evaluation of programs. The evaluation of the programs produced is performed within a simulated environment. This, along with distributed evaluation, allows the whole evolutionary process to be significantly speeded up.

The evolutionary system consists of an interpreter integrated with a robot environment and a GP engine. The integration is required to allow the evaluation of the programs produced by the GP. The system can accept any number of robots running either heterogeneous or homogeneous programs. However, at present it uses three robots each running the same program. In the following sections, the workings of the robot environment and the interpreter are explained.

### 5.5.1 Interpreter

The interpreter executes programs that are composed of any number of the three basic building blocks. The basic building blocks are:

- procedures (Automatically Defined Functions (ADFs)),
- threads,
- main bodies.

Each block in a program has a unique identification number associated with it. A procedure is a piece of code which can have arguments passed into it and which returns a single value. The calling of a procedure is indicated by ***#n***, where **n** is the block id associated with the procedure. A thread is a piece of code which does not need to be instigated (allowing for multi-threaded programming). A thread starts to run from the inception of the program and is repeated until the program ends. The main block is the piece of code which controls the flow of the program. It starts before any threads and repeats as long as the simulation lasts. The combining of these blocks in an ordered sequence (or list) gives rise to what is termed a system. A system can consist of any number of procedural and thread blocks but must contain only one main block, which should be located at the end of the system list. The programs used in this work do not make use of the thread block type. As is common practice in GPs, the interpreter processes programs using prefix notation. Figure 5.2 gives a simple example of a program consisting of 2 ADFs a thread and a main body. The general structure of this figure is common to all programs. This structure is broken down as follows: after each block type is the block id number (except for main) e.g. *procedure1* (where 1 is the block id and procedure is the block type), which is then followed by a set of braces containing two values. The first value defines the number of input parameters to the block and the second the number of return parameters from the block. The value following these braces identifies the set of functions and terminals the block is permitted to use (for GP engine use only). The contents of the [] brackets define the code associated with the block. In the case of the system it defines the program. The interpreter is stack based so all parameters and actions are passed via or performed using the stack. Any parameters passed into a block are accessed using the ***@n*** operator where **n** indicates which parameter is referenced.

```
system[
   procedure1{1,1}2[ (+ 10 (#2 @1 (/ 20 @1)))]_Proc

   procedure2{2,1}3[ (* @1 (- 10 @2))]_Proc

   thread3{0,1}6[ show("Still running ")]_Thread

   main{0,-1}8[ show (+ (#1 62) 7)]_Main
 ]Sys
```

Figure 5.2 Example of program containing all possible block types.

Depending on the scheduling of the processes, the output from the program in Figure 5.2 will change (due to the presence of the thread). However, if we assume the following scheduling order: main, thread, main, procedure1, procedure2, thread, procedure1, thread, main. The following output will be generated:

Still running Still running Still running 607

As it can be seen the "Still running" text is outputted 3 times by the thread (corresponding to the number of times the thread was run) and the value 607 is output by the main block. The reason the value 607 is at the end and is only outputted once is that in the time it takes to execute the main block once the thread is able to execute three times i.e. the thread runs three times faster than the main block (this is due to it small size and lack of procedure calls).

### 5.5.2 Simulated environment

The execution of a program by the robots within the simulator is run asynchronously, which allows for the robots to move while their program is being executed and also move in different orders, allowing for asynchronous communication. The programs return a single real value (unless otherwise stated) which is used to define the turning angle of the robots. This turning angle is added to the robots current destination heading value. The robot continues to turn in the indicated direction until its current heading is equal to the destination heading then travels in a straight line. The current heading and destination heading of the robot are initially equal. To ensure that the value returned by the programs are always between 0 and 1 the function in Figure 5.3 is always used (where values < 0.5 indicate an anti-clockwise turn and values greater a clockwise turn.)

```
if (Heading>1.0) // Ensure rotation value in valid range
  Heading=1.0/Heading;
else if (Heading < 0.0)
     Heading=1.0+(1.0/(Heading-1.0));
Heading=fabs(fmod(Heading,Limit));

if (Heading >= 0.5) // Convert rotation value to angle in radians
  {
  if (Heading!=0.5)
    DirectionOfTurn=Clockwise;
  else
     DirectionOfTurn= DirectionOfTurn;
  Heading=Heading-0.5;
  }
else
    if (Heading!=0.0)
      DirectionOfTurn=Anti-clockwise;
    else
        DirectionOfTurn= DirectionOfTurn;
Heading=(((Heading*2.0)*MaxTurnAngle)*(2.0*PI))/360.0;
```

Figure 5.3 Code to ensure that a value between 0.0 and 1.0 is always used for heading.

The robots move a given distance on each interpreter cycle or t-state. Whilst these movements are occurring the robot programs are continuously being evaluated, which allows for sensor reading and communication to change during the duration of a programs execution. This also serves to limit the size of the programs, since the larger the program, the fewer times it will run, the less responsive it will be and the more it will rely on outdated sensor readings and communications. The use of the energy consumption as the terminating criteria defines a fixed amount of time for each program to run and illustrates its worth. This is in contrast to the methods adopted by other researchers, in which a fixed number of runs of each program are used. In their work, each program is run to completion then the robot is advanced a step in the indicated direction [124,125,137]. Such methods rely on the use of static sensor readings and communication and assume all programs take the same amount of time to run regardless of their length. Using the dynamic method employed here means programs such as:

IF ($Ty_i$ > $Ty_{i+j}$) THEN
  go straight;
ELSE
  turn left 20 degrees

actually perform a function. If **Ty** is data received from a robot concerning its y co-ordinate, $\mathbf{Ty_i}$ means data received at time **i** and $\mathbf{Ty_{i+j}}$ means data received **j** time units later. In this case the program translates as follows (assuming continuous transmission

of co-ordinates): if the robot from whom communication has just been received is further away along the y-axis than the robot whose communication was previously received, then move straight otherwise turn left 20°.

To achieve this asynchronously the interpreter time slices the programs, giving rise to pseudo parallel execution and the simulator allows for a random movement order of the robots. The interpreter is set up such that at the end of each fetch cycle the robots are allowed to move and to communicate any information to those in range (the interaction between the simulator, interpreter and GP engine is shown in Figure 5.4). The number of fetch cycles required before an instruction is executed varies from 1-6 depending on the number of parameters it uses and its classification.

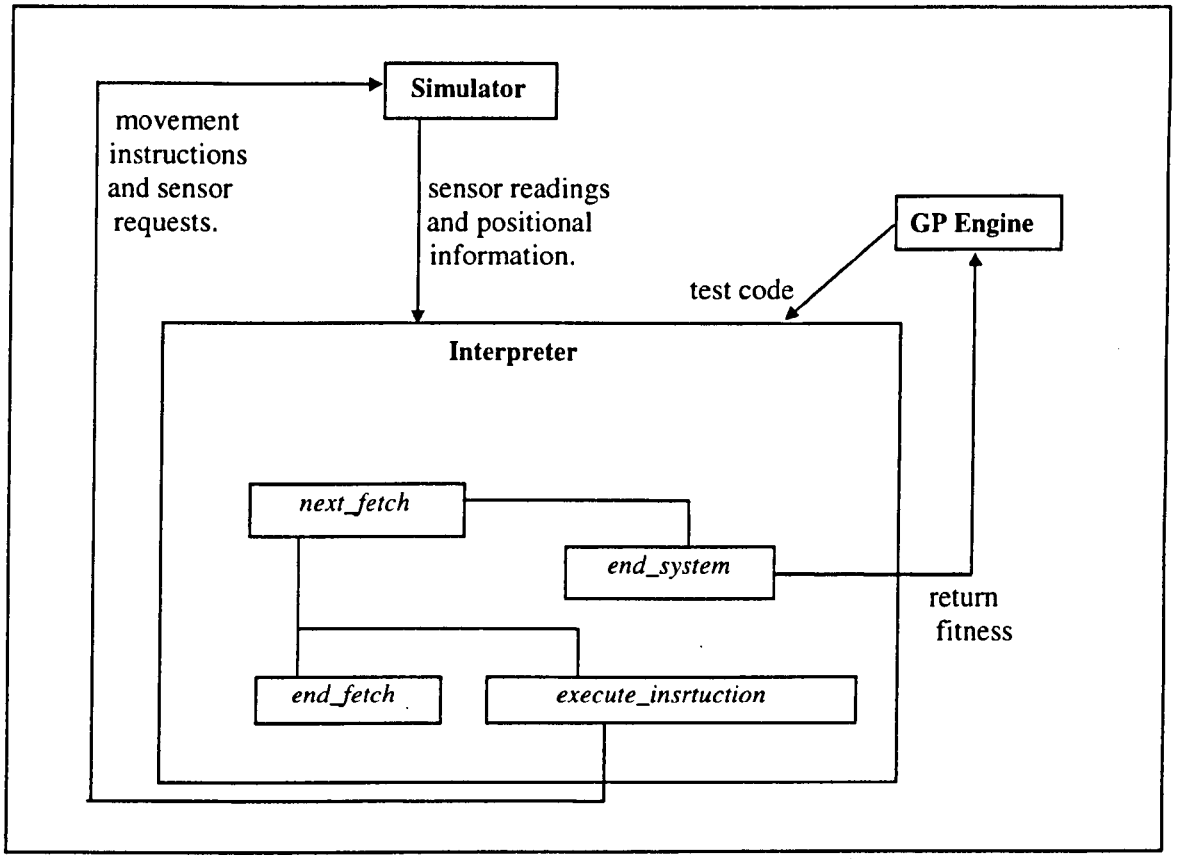


Figure 5.4 Interaction between simulator, interpreter and GP engine.

Table 5.2 shows a time-annotated execution of three different programs using the time slicing method adopted in the interpreter run for 26 time steps. The maximum time slice

was 6 t-states, the resultant order of execution of the programs was: **a**,**c**,**b**,**c**,**b**,**a**. In this interval, program **a** and **b** run to completion twice and program **c** once.

program a:
```
system[ main{0,-1}0[ + 10.6 * 8.0 1.5]_Main ]_Sys
```
program b:
```
system[ main{0,-1}0[ * 50.3 0.5]_Main ]_Sys
```
program c:
```
system[
        main{1,-1}0[
           IF>= 10.0 * @1 20.0
                     Then 30.0
                     Else / 10.0 @1
            ]_Main
    ]_Sys
```

| TIME | PROGRAM | ACTION |
|---|---|---|
| 1 | a | get + |
| 2 | a | get 10.6 |
| 3 | a | get * |
| 4 | a | get 8.0 |
| 5 | a | get 1.5; do multiplication |
| 6 | a | do addition; return value; restart program |
| 7 | c | get IF>= |
| 8 | c | get 10.0 |
| 9 | c | get * |
| 10 | c | get @1 |
| 11 | c | get 20.0; do multiplication |
| 12 | c | do comparison; swap out |
| 13 | b | get * |
| 14 | b | get 50.3 |
| 15 | b | get 0.5; do multiplication; return value; restart program |
| 16 | c | move to true section of if statement |
| 17 | c | get 30.0; return value; restart program |
| 18 | b | get * |
| 19 | b | get 50.3 |
| 20 | b | get 0.5; do multiplication; return value; restart program |
| 21 | a | get + |
| 22 | a | get 10.6 |
| 23 | a | get * |
| 24 | a | get 8.0 |
| 25 | a | get 1.5; do multiplication |
| 26 | a | do addition; return value; restart program |

Table 5.2 The execution stages of three programs in a time sliced interpreter using quantum length of six cycles.

### 5.5.3 Distributed evaluation

The evaluations of the fitness of some of the tasks are carried out using a distributed evaluation system. In this approach a number of satellite machines can enter and leave the evaluation process at will without altering the robustness of the system. The server program is started up on a single computer, this program performs all the genetic and evolutionary functions. As well as this, it creates a list of pending evaluation jobs which other machines can gain access to. The server goes into competition with the other machines to acquire these jobs to evaluate. Once the server processes its last job it checks to see what jobs results have been returned and which ones are outstanding. At this point, it suspends the access to the job list causing other competing idle processes to sleep. It now evaluates all outstanding results assuming that the processes with one of these jobs are no longer participating. Once all the results are computed the server generates the next population, empties the old job and fitness lists creating a new job list at the same time. Due to the distributed nature of the system, process co-ordination is required to avoid race conditions. This is achieved by using a simple inter-process communication method based around the idea of semaphores. Here a binary semaphore is approximated by using files, which are locked and unlocked. Applying the *lock* command to a file is similar to the semaphore *down* command, this allows only a single process to gain access to the file and all other processes are put to sleep. Rather than the binary semaphore employed here, counting semaphores are possible by placing a counter within the file, which determines how many processes can enter a critical region. The *unlock* command is similar to the semaphore *up* command, as it releases the lock on a file and causes all processes seeking it to be woken up and allowed to try again to gain the lock on the file. These distributed semaphores are used to restrict access to the job and fitness lists critical regions. This restriction allows only one process at a time to choose an available job, ensuring that only one process is assigned to evaluate a job. Also it allows for the orderly returning of results by the satellite machines, which ensures that a process is not interrupted by another while it is mid-way through writing it results to the fitness file producing legible results.

Using this system, the satellite machines just need to be made aware of which files to lock to enter each critical region. This is supplied by a common *yellow pages* file. They then wait for the list of new pending jobs to be posted and then try to get one. On getting one they flag that the job has been taken and then go about evaluating it using the same simulator, interpreter and task settings as the server process. Once they have processed the job they gain access to the fitness list and write the results there as long as it has not been emptied since they got the job. The use of this method means that the server process does not need to do any additional processing when a process wants to join or leave the processing team, in fact it assumes that no other machine is aiding it and only knows otherwise when it goes to collect the results from the fitness file.

## 5.6 SUMMARY

This chapter has presented the tools required to perform the genetic programming experiments. A genetic programming engine and its associated parameters were presented together with an integrated simulator and interpreter. The combination of these allows evolved programs to be evaluated in a simulated environment and then improved by evolutionary pressures. The use of asynchronous modelling in the simulation and interpreter gives rise to more realistic test environments than those reported in the literature.

# Chapter 6

# TASK II: MEETING UP OF MULTIPLE ROBOTS

## 6.1 INTRODUCTION

The aim of the work presented in this chapter is to discover whether programs evolved to control robots in multi-robot environments can use and benefit from the presence of communicated information. To accomplish this, a co-operative task was devised which required that the members of a multi-robot environment stay as close to each other as possible for as long as possible (within a bounded environment), whilst minimising the number of collisions between each other. Each of the robots is equipped with the same initial energy level and any attempted movement consumes a fixed amount of this energy. The task ends when all of the robots have run out of energy. Three robots, each of which runs the same program, are used within a 100x120 grid environment. The robots continuously broadcast the information they are able to communicate. Several forms of information content are investigated as well as the effect of communication range. It is the job of the evolved programs to decide when to receive information and what to do with it. In section 6.2 the implementation details of the task are covered, section 6.3 describes the functions and terminals used in this work, section 6.4 presents a series of three incremental experiments designed to discover the extent to which communication can be used by the evolutionary process and the conclusions drawn from these experiments are presented in section 6.5.

## 6.2 TASK OVERVIEW AND IMPLEMENTATION

The experimental parameters used in the task are outlined in section 6.2.1. Section 6.2.2 highlights key features of the task and indicates how they are relevant to the

investigation of the systems ability to use and benefit from the presence of communicated information. Section 6.2.3, presents in detail the task and its implementation.

### 6.2.1 Experimental parameters

Four task specific parameters are defined for these experiments. These are:

- Number of robots 3
- Test suite size and number of test chosen *Part I:* size 2; chose All
  *Part II*: size 6; chose 2
- Number of independent test runs. 4
- Duration before communicated information timed-out. 30

The values of these parameters were determined as a result of a set of preparatory experiments. The factors that influenced the levels of these parameters are presented below.

Three robots were used in this task since this number was the minimum required to implement the task.

The test suite size and number of tests chosen were designed to encourage the evolution of robust controllers (see section 6.2.3 for more details). All the tests in Part I of the test procedure had to be used with two basic controllers available. These encapsulated the general nature of relevant motions applicable to the testing regime, one of the controllers offered fairly predictable behaviour and the other unpredictable. Part II of the test procedure required that two tests be chosen randomly from a suite of 6. This was made necessary owing to time constraints, that is not all 6 test cases could be evaluated at the same time. This random choice was found to encourage robustness. The tests were designed such that each encouraged a different facet of the controllers such

as, what to do when close or far away from each other, what to do when the robots can not see each other etc.

Time constraints did not permit for any more than 4 independent test runs.

The duration before communicated information timed-out was set to 30. This was used as away of simulating relevancy of communicated information. The level was set such that the average program could run two to three times. If the controller relies on this mechanism the system will be using outdated communicated information. This level was determined experimentally. It also avoids continuous responding to last received data.

### 6.2.2 Task overview

The task, to be undertaken is for the robots to meet up within a bounded and continuous broadcast environment. The construction of the task and its evaluation method was undertaken so as to promote the study of some of the issues and properties of importance within robot communication systems, some of which can be generalised beyond the task, others which will be specific to the rendezvous/pursuit class of tasks. The possible addressable issues and properties encapsulated by the task are:

- multiple avenues of task accomplishment
- deficiency exploitation and rectification
- competing strategies
- unstructured and unconstrained usage of communicated information
- accuracy and timeliness
- multi-phase testing
- is there a benefit in the act of communication

By allowing for multiple and distinct potential avenues for solving the task, the property of dual accomplishment is incorporated within the task. Here there are two

distinct avenues for solving the problem, one using visual operators and the other using communication. The traditional view is that for communication to be adopted in an evolutionary based tool effectively there should be only one viable avenue for solving the problem (i.e. using communication). Here, by providing non-biased multiple and distinct avenues for solving the problem this view can be investigated. In addition, by using various forms of communication the circumstances under which this bias is most required can be identified. The results of this will be of general relevance in the designing of tasks in which the evolved use of communication is a factor.

Each of the alternative methods for solving the problem have innate advantages and disadvantages associated with them. The advantages promote their use by the evolutionary system and the disadvantages potentially demote their use. By allowing these methods to co-evolve and compete, any exploitable deficiencies in them allow other methods opportunities to dominate them or work alongside them (this is the idea of deficiency exploitation and rectification). The relevance of the deficiencies and the level of exploitation and collaboration between methods are to be determined by the evolutionary process. However, each method is capable of performing or offering some benefit to the performance of the task. Some of the general deficiencies or restrictions on operation are vision based operators incapable of resolving distance; restricted angle of view, communication may require additional processing, high degree of accuracy or have a restricted range of operation. The presence of these deficiencies/restrictions has a threefold effect. Firstly, they allow for faults, deficiencies or under specifications to be identified within a method. Secondly if such faults, deficiencies are inherent to the method it offers a measure (by way of evolutionary survival) of the potential drawback it causes the control system as a whole and thirdly, whether or how easily they can be patched as well as offering some indication of how to do so.

The task contains competing strategies, these are, getting close together, whilst simultaneously avoiding colliding with each other. Each of the strategies are co-evolved, so it requires the evolutionary process to give each the appropriate weighting as well as develop the optimum collective strategy. The boundedness of the environment promotes

the need for an effective collision resolution strategy. Within each of the strategies the need and use for and of communicated information is potentially distinct, as such the evolutionary process must be able to apply the information content in a flexible and appropriate manner.

In the task, simple functional building blocks are used. This gives the evolutionary process all the freedom to develop the structure of a controller in the way that best suits it. This unstructured and unconstrained development of programs allows maximum freedom in the placing and usage of communicated information within evolved controllers and the strategies they contain.

By providing communication functions whose information content is primarily to do with distance or positional measures, the idea of accuracy of received information can be investigated. Since the robots are continuously moving, any information received by another robot is already out of date. This places pressures on the size of programs and where and how often they use communicated information. Timeliness is addressed via the use of various communication ranges and communication time-out for received information (the use of different communication ranges also allow for investigations into how range effects the usage of communicated information). The study of these two issues allows for decisions on the benefit of centralised and distributed control and communication to be drawn. If timeliness and accuracy are not an issue then centralised control and communication systems are viable. If however, one or both of them are of significance then distributed control and communication is the only feasible option.

The task uses a two-phase evaluation system for evolved programs to encourage robustness. Part I was implemented to avoid the development of trivial solutions which effectively encode a prescribed meeting place in them, obviating the need for the use of communication or visual operators. Here a single robot (using the program to be evaluated) is placed in an environment containing two other robots executing different programs from it, which in turn make no use of any communicated information. However, as well as avoiding trivial meet at (x,y) solutions, this part of the evaluation

process also plays the added roles of simulating interactions with uncooperative robots, circumstances where communication receive abilities have failed in two robots or the interaction between robots running distinct controllers, where one is less adept than the other at the task. So by succeeding in this part of the evaluation process a controller must posses either a high degree of generality or a high degree of robustness to faults or breakdowns in communication. However, this general level of fault tolerance is not the only robustness required. A high degree of robustness and efficiency is also wanted for cases where there are minimal faults in the environment and all or most of the individuals are being co-operative. This is encouraged by part II of the evaluation process in which all three robots run the test program. The robustness is sought here by introducing noise into the evaluation process. The combined effect of the evaluation process is for the development of general, fault-tolerant and robust controllers.

This task sets the platform from which a series of incremental tasks can be carried out to determine if it is the presence of communicated information or the act of communication that is being exploited by the evolutionary process (results from here indicate there is large degree of relevance placed on when exactly to use the received information, manifested in the form of behavioural triggers, implying that even within continuous broadcast systems there are times when communication is not required). This is relevant since if there is a cost associated with communication (which in most cases there is, but which is not simulated here) then by minimising the times or the rate at which it takes place then more reliable and cost effective use will be made of communication. This however is not the topic of this chapter but is instead considered in the next chapter (chapter 7). This chapter concentrates on whether a role can be found for communicated information and the implementation details of the task adopted for this investigation follow.

### 6.2.3 Task implementation

In order to stop the evolution strategies which encode a specified location as a meeting place (which circumvent the need for communication) as well as attempting to improve

the robustness of programs a two stage test program is used. In part I a single robot employs the program to be evaluated in an environment containing two drone robots (drones here are considered as robots, which are not under the control of the program being evaluated). These drones both use a pre-specified program to define their movement. Part II uses all three of the robots each running the test program. The test cases used are changed randomly every generation. Part I will be used to help avoid the meet_at_(x,y) strategy, since the drones will have no knowledge of any rendezvous point encoded into the evolved program and part II to encourage robustness and diversity.

Each of these two parts consisted of two test scenarios, in which the position and orientation of the robots would be different. Further, in the case of part I test scenarios, a different program was executed in each case by the drones (these can be seen in Appendix A). The test scenarios were constructed to see how the programs would perform when the robots are close to each other, when they are far away from each other and when there are pending collisions. The programs executed by the drones were designed not to respond to communication and to exhibit either jittery behaviour (scenario i) or constant smooth behaviour (scenario ii.) These were used to encourage the evolution of controllers capable of approaching both randomly and regularly moving robots. The test scenarios of part II were chosen from a set of six cases.

A safety zone is defined around each robot, so robots may be penalised if they move to close to each other. In addition, a buffer zone is used as a way of encouraging the robots to stay close. Inside this buffer zone, the distance between the robots for evaluation purposes is taken to be zero. The safety zone is totally encompassed by the buffer zone.

The fitness of a program is defined by two constraints:

1. the average distance between the robots throughout the testing process
2. the percentage of time spent without colliding

A percentage of the collision value (constraint 2) is also added in to the first constraint to compensate for collisions with the environment boundary. Constraint 1 is obtained by calculating the distances between each of the robots and the other members each time the interpreter swaps out a program (see section 5.4). This value is tallied for each test case and averaged over the number of robots. The final value is obtained by averaging each of the test cases. The second constraint is obtained by counting the number of interpreter cycles a robot moves for, and the number of collisions it is involved in during that period. These values are averaged for each of the robots to obtain the performance for a test, then averaged over each test to get the final value. By minimising the fitness vector, the evolutionary process will be able to produce programs increasingly more capable of achieving the task. The algorithms used to determine the fitness of a program are shown Appendix A.

A tolerance level is set so that only effective controllers are logged. The tolerance vector level for constraint 1 is set at 500 and wildcard for constraint 2. This means that all programs with constraint 1 value of less than 500 will be logged. This level was chosen because visual inspection of some evolved controllers showed that fitness values of about 600 produced controllers which were beginning to employ some meaningful and potentially useful strategies. However, by around the 500s these strategies were starting to become quite effective. Those in the 400s prove to be effective and in some cases very effective. Any programs with values in the 300s were extremely effective. The controllers employ an effective collision resolution strategy, which is compatible with its co-ordination method, when the value of its constraint 2 value is less than 35. Using these values the previously logged controllers can have their performance categorised and, where appropriate, further scrutinised.

## 6.3 FUNCTIONS AND TERMINALS

The terms function and terminal, with respect to genetic programming, are defined as follows in this thesis. Function:- genetic element which has one or more input

parameters; Terminal:- genetic element that has no input parameters. The various experiments reported here investigate the effect of the presence of different kinds of communicated information, in all these experiments a common set of standard functions and terminals are used. Table 6.1 lists these functions and terminals in conjunction with a description of how they work. The reasons for choosing these functions and terminals are as follows:

- To allow for flexible processing of the received communicated information. This was to be achieved via the provision of the mathematical operators. This also creates a need for the evolutionary process to determine order dependency between its controllers' strategies if they are used as sequencing operators.
- To allow a degree of unpredictability to be incorporated into the controllers strategies. For this, random and conditional operations were provided.
- To encourage smaller controllers, through the ability of producing compact expressions.
- To allow the evolutionary process to determine how best the received communicated information should be interpreted. Various re-interpretation operators were provided for this purpose (e.g. Rinv, Null, Flip, SH).

Six additional operators are utilised for the receipt of communication. Each of these operators takes no parameters and returns a single real number, which indicates the received information communicated at the time of use. A list of these operators is given in Table 6.2 along with a brief description of their operation. These operators are employed in the various experiments described in this section and a more detailed explanation of their workings can be found in the sections where they are used. The table also contains a list of available visual operators and a description of their operations. Table 6.3 defines the actual contents of the function terminal sets used throughout this chapter together with their arbitrary identification values and shows the experiments in which they are used.

| FUNCTIONS & TERMINALS | DESCRIPTION |
|---|---|
| IFGE **a,b,c,d** | if **a** greater or equal to **b** then return **c** else return **d**. |
| PA **a,b** | probability action operator, generate a random number if this value is less than **a** then return **b** otherwise return 0.0. |
| +;-;* **a,b** | standard arithmetic operations performed using **a** and **b** and result returned. |
| / **a,b** | return the result of **a** divided by **b**, in the case where **b** is zero then it returns **a**. |
| Bk **a** | add **a** on to the reverse distance tally, return 0.0. |
| Max **a,b** | return the largest value out of **a** and **b**. |
| Min **a,b** | return the smallest value out of **a** and **b**. |
| RInv **a** | invert rotation amount, if **a** is in the range of 0..1 then subtract the angle it represent from 180 degrees, otherwise return **a** unchanged. |
| Null **a** | return 0.0 if **a** is close to 0.0, 0.5 or 1.0 otherwise return **a** unchanged. **a** must be in the range of 0..1 for it to be changed. |
| Flip **a** | change rotation direction indicated by **a**, if it is in the range of 0..1 otherwise leave value unaltered. The rotation direction is changed by adding 0.5 to values in the range 0.0 >= **a** < 0.5 and 0.5 is subtracted from values in the range 0.5 >= **a** <= 1.0. |
| SH **a** | if **a** is in the range of 0..1 then calculate the angle the robot needs to turn through so that it heading is **a**. The direction of rotation is dependent on that indicated by **a**. The angle to rotate through can not exceed 180 degrees. If **a** is not in the valid range then it is returned unchanged. |
| (_) | no operation, return 0.0; |
| NoMv | collision flag, returns 0.0 if no collision otherwise returns 1.0. |
| r? | random real value. |
| r | real constant. |

Table 6.1 Standard functions and terminals available to GP.

| FUNCTIONS & TERMINALS | DESCRIPTION |
|---|---|
| GSCom | get simple communication value. |
| Gdx | get dx value. |
| Gdy | get dy value. |
| GRdx | get relative dx value. |
| GRdy | get relative dy value. |
| GVI | get light communication |
| OnLSL **a,b** | On sensing light in the left visual sector perform action **a** otherwise perform **b**. |
| OnLSC **a,b** | On sensing light in the central visual sector perform action **a** otherwise perform **b**. |
| OnLSR **a,b** | On sensing light in the right visual sector perform action **a** otherwise perform **b**. |

Table 6.2 Communication receive functions and visual operators available to the GP.

| SET ID | EXPERIMENTS USED IN | FUNCTIONS AND TERMINALS |
|---|---|---|
| 6 | III | {IFGE, PA ,+, - , *, /, Bk, Max, Min, Rinv, Null, Flip, SH, (_), NoMv, r?, r, GVI} |
| 7 | III | {IFGE, PA, +, -, *, /, Bk, Max, Min, Rinv, Null, Flip, SH, (_), NoMv, r?, r, Gdx, Gdy} |
| 8 | III | {IFGE, PA, +, -, *, /, Bk, Max, Min, Rinv, Null, Flip, SH, (_), NoMv, r?, r, GRdx, GRdy} |
| 11 | I,II | {IFGE, PA, +, -, *, /, Bk, Max, Min, Rinv, Null, Flip, SH, (_), NoMv, r?, r, OnLSC, OnLSR, OnLSL, GSCom} |
| 12 | I,II | {IFGE, PA, +, -, *, /, Bk, Max, Min, Rinv, Null, Flip, SH,(_), NoMv, r?, r, OnLSC, OnLSR, OnLSL, Gdx, Gdy} |

Table 6.3 Functions and terminal sets and the experiments in which they are used.

## 6.4 EXPERIMENTS AND ANALYSIS

The robots have a visual ability, which is global in nature, a view implemented by way of light detectors. The robots have no way of indicating distance, just relative position within the visual angle. This is achieved by separating the visual field into three equally

sized sectors: left, centre and right. The communication present in the environment has two basic forms of use: direct and indirect. In the direct case, the received communicated information is applied in an unaltered manner to some ends whereas in the indirect form it is processed as part of some equation or computational structure.

### 6.4.1 Overview of evolved controller types

This section introduces the kind of controllers that emerged in the initial experiments.

The evolutionary process was able to evolve controllers capable of resolving collisions and staying close together. To do this it had to resolve and prioritise competing strategies. Both the meet up strategy and the sub task of collision resolution were simultaneously being evolved. A degree of mutual exclusion exists between these two strategies, in that one is trying to get the robots to touch and the other is trying to stop them from doing so. Due to the elitist reproduction method and the mating pool bias in favour of the meet up strategy (constraint 1), it is given evolutionary priority over the collision avoidance strategy (constraint 2). The effect this prioritisation has on the collision resolution strategy is to say that in order for any improvements in this pool of strategies to be long lived and/or more influential it is best to combine them into the meet up strategy. However, it should be noted that these are just evolutionary pressures and not concrete specifications, so it is possible for them be ignored. If the evolutionary processes is capable of producing controllers with only a single strategy in them which performs as well as the combined case it will. Such controllers were produced in these experiments. In these cases it was usually the collision resolution strategy that was left out or had a token presence. The best results were always obtained when both strategies were present and were self-complementing.

The initial successful controllers evolved had a simple general form, which was to turn in a circle until a robot is spotted in a given visual sector, then carry on along the same heading. This can be expressed as in Figure 6.1a and a visual representation of such a path can be seen in Figure 6.1b.

The (_) operator can be replaced by any operator or combination of operators that returns either zero or one. The angle turned through has to be relatively small otherwise the chances of spotting a robot will be quite slim. Turns of between 15 and 40 degrees perform best. In addition, for long programs to perform well their angle has to be towards or beneath the low end of this range. The experiments showed that over time shorter programs were found to predominate. These straight liner solutions were easy to find so they tended to dominate the population initially. However, this was then followed by the slow evolution of snaking controllers, which instead of having a blank first parameter replace it with a turn opposite in direction to the second parameter. The general form of these was to: turn in a circle until a robot is spotted in a given visual sector then turn in the opposite direction. Figure 6.2a shows the general form of this and Figure 6.2b shows the general path produced by such a program, which gives rise to its name (snaker).

```
system[
    main{0,-1}0[
         (OnSLR          // if light_sensed_on_right then
             (_),        //   return 0° turning angle
             0.57895     // otherwise return a turning angle of 14.21° clockwise
         )
     ]_Main
 ]_Sys
```

Figure 6.1a General form of straight liner.

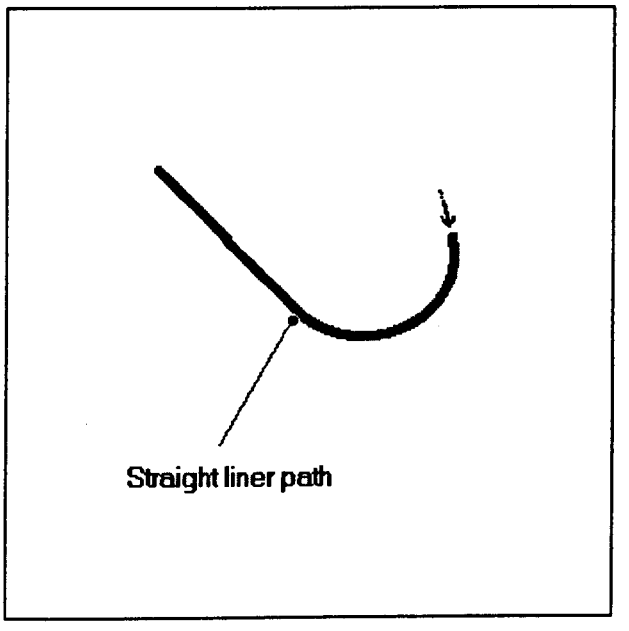



Figure 6.1b General path of a straight liner.

```
system[
    main{0,-1}0[
        (OnSLL          // if light_sensed_on_left then
            0.12572     //   return a turning angle of 22.63° anti-clockwise
            0.54824     // otherwise return a turning angle of 8.68° clockwise
        )
      ]_Main
]_Sys
```

Figure 6.2a General form of a snaker.

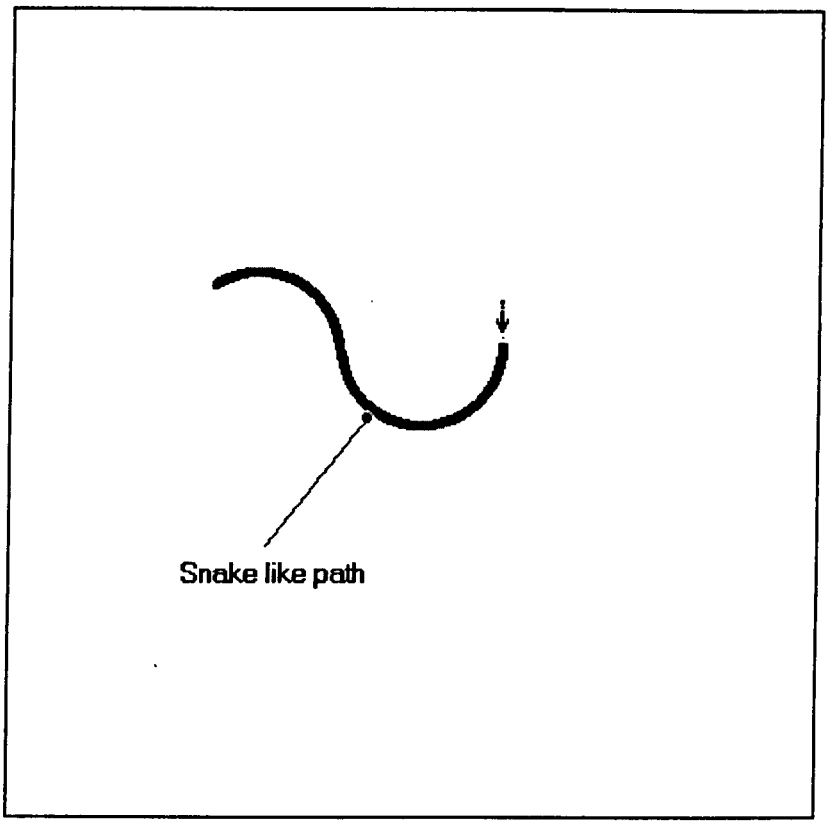



Figure 6.2b General path of a snaker.

The angles in these snaking controllers had to be within the same range as the straight liners in order to produce effective performance. However, there also had to be a degree of asymmetry in the turn angles. The improved performance offered by the snaking behaviour was due to the fact that the target robot was moving, so turning in both directions increased the chances of keeping the target in the visual sector used (since if the target leaves this sector the robot has to do almost a complete 360° turn over a large distance in order to bring it back into view again, which wastes time and leads to worse performance). Another method was also employed to overcome the problem of losing sight of the target robot. This required the use of random or large turns when the target

robot changes visual sector. An example of this can be seen in Figure 6.3. These alternative controllers also came as snakers and straight liners and only worked well if the visual sectors tested for were adjacent to each other and the jerk angle they turned through was able to bring the target back into view.

```
system[
    main{0,-1}0[
        (OnSLL                          // if light_sensed_on_left then
            (OnSLC   r? 0.12572)        //   if light_sensed_in_center then
                                        //     turn randomly
                                        //   otherwise turn 22.63° anti-clockwise.
            0.54824                     //  otherwise return a turning angle of 8.68°
                                        // clockwise
        )
    ]_Main
]_Sys
```

Figure 6.3 General form of a jerk angle snaker.

The best snaking controllers were obtained when the central visual sector was used as the primary sector. The snaking controllers emerged in small numbers typically after 30 or so generations, then, after a further 70 generations they tended to have a significant hold on the population.

It took longer for the controllers to start to evolve their collision resolution strategy as compared to their ability to approach each other, but, once the collision avoidance started to appear, it was quickly incorporated into the controllers. The initial successful collision avoidance strategy was to reverse continuously a given amount, which depended on the size of the program utilising it, such that the larger the program the larger the reverse distance had to be. However, to accomplish the main task the distance reversed through should not over hamper the forward motion towards the other robots, so these controllers evolved such that they reversed enough just to allow the robots to turn through a small angle. These controllers avoided collisions using a series of small reverses as opposed to one large one, Figure 6.4 shows such a controller. An improvement on the controllers of Figure 6.4 came by using the NoMv operator to indicate when to reverse. In these cases, one large reversal could be used to resolve collisions. When collisions occurred between robots, those controllers which used different reverse amounts for each visual sector or those which used random reversal amounts performed best. General examples of these can be seen in Figures 6.5 to 6.7.

```
system[
    main{0,-1}0[
        (OnSLR
            (_)
            (+ (Bk 0.34872) 0.51293)
        )
    ]_Main
]_Sys
```

Figure 6.4 Simple sensor dependent collision resolution strategy.

```
system[
    main{0,-1}0[
        (OnSLR
            (_)
            (+ (Bk (NoMv)) 0.51293)
        )
    ]_Main
]_Sys
```

Figure 6.5 Sensor and collision dependent reversal strategy.

```
system[
    main{0,-1}0[
        (OnSLR
            (Bk 0.12739)
            (+ (Bk 0.46175) 0.51293)
        )
    ]_Main
]_Sys
```

Figure 6.6 Multiple sensor dependent reversal.

```
system[
    main{0,-1}0[
        (OnSLR
            (_)
            (+
                (Bk (* (NoMv) r?))
                0.51293
            )
        )
    ]_Main
]_Sys
```

Figure 6.7 Sensor dependent random reversal on collision.

These types of controllers only reversed when certain visual situations arose and so tended to work well only when the robots were facing each other and hence were not very robust. The next form of collision resolution strategy that emerged employed the reversal operator regardless of visual situation, an example of this can be seen in Figure 6.8. These forms of controllers in some cases also utilised additional visual dependent reversals. Their performance was best when the NoMv operator was used to activate the reversal.

```
system[
    main{0,-1}0[
        (/
            (OnLSR (_) 0.51293)
            (Bk 0.13269)
        )
    ]_Main
]_Sys
```

Figure 6.8 Continuous reversal.

The incorporation of the receive communication operators into the frameworks of the general controllers presented in this section is covered in sections 6.4.2 to 6.4.4. This incorporation lead in most cases to interesting approximations of these controllers. The reason for this is the varying information content available in the environment and the range over which it is communicated.

The following sections report in more detail the results obtained from a series of experiments. However, although the type and range of communicated information used varied between experiments the processing time remained relatively constant. To complete a single run took around $2^1/_2$ days of computational time. The experiments were carried out on a HP-700 series workstation (no distributed evaluation was used here), using four sets of test runs in each case. In each of the four sets of test runs, a distinct set of initial seeds was used for the random streams in the GP engine.

### 6.4.2 Experiment I

*Aim:* To determine the role locally communicated information content will play in the presence of visual co-ordination operators as well as to determine if the role it plays is dependent on the information content.

In this experiment, the transmission range of the robots is limited to a 35 unit radius about their centre. The information is communicated is both involuntary and continuous. Two forms of information are utilised: a simple constant, and the x, y position of the robot.

**Simple constant:**

This entails the transmission of the value 1.0. The operator GSCom is used for this. This was provided to see if the communicated information present in the environment could be useful when only the absolute minimum information is conveyed.

**Displacement x, y:**

This requires two communication operators, one to handle the x co-ordinate (Gdx) and the other to handle the y co-ordinate (Gdy). The robots transmit their x and y position, whereupon the receiver interprets the receipt of either of these as a displacement from the transmitting robot along the appropriate axis. A negative displacement is indicated by a value in the range 0.0 to 0.5 and a positive displacement as a value in the range of 0.5 to 1.0, where the values 0.499 and 1.0 imply the robots are at the edge of the communication range and 0.0 and 0.5 imply that they are occupying the same point.

### 6.4.2.1 Results of experiment I

The main use of communicated information here was in the reversal strategies, examples of which can be seen in Figures 6.9 and 6.11. Figure 6.9 shows a controller using sensor dependent collision resolution and Figure 6.11 shows a controller using a multiple sensor and communication range dependent strategy. The use of such communication is one way of ensuring that the robots are kept some distance away from each other. This has the obvious effect of reducing the number of collisions and the additional advantage of giving extra time to cope with any sudden or abrupt direction changes made by other robots. Being far away made it harder for the other robots to move too far away from the follower's viewing angle. In this respect, the use of simple communication performed best. The most common use of communication in the co-ordination strategy was as an action trigger. Here, the limited range of communication was exploited as a means to instigate alternative actions, which would occur when two or more robots came within communication range of each other.

**Simple communication:**

The average time taken to find the first tolerable controller was 9 generations. The average fitness of these controllers was [479 115], where 479 is the value of constraint one and 115 of constraint two. The optimisation of the controllers continued for on average 150 generations. The average fitness for the best controller was [375 22].

An early evolved program given in Figure 6.9 shows multiple usages of communicated information within the same controller which is highlighted in bold. The first is to switch the controller type and the second is used as part of the collision resolution strategy.

```
system[
    main{0,-1}11[
        (/
            (OnLSC
                (OnLSR
                    r?
                    (*
                        (* 0.08979 r?)
                        (OnLSL (GSCom) (+ r? 0.50358))
                    )
                )
                0.54164
            )
            (Bk
                (OnLSC
                    (OnLSR
                        (OnLSR
                            (* 0.90984 0.42496)
                            (OnLSR
                                r?
                                (/
                                    (OnLSC r? 0.53969)
                                    (Bk (OnLSC r? (NoMv)))
                                )
                            )
                        )
                        (Bk (GSCom))
                    )
                    (NoMv)
                )
            )
        )
    ]_Main
]_Sys
```

Figure 6.9 Controller exhibiting detrimental use of communication.

The program is primarily a snaker but becomes a straight liner when a target is first detected by the central visual sensor then one is detected on the left and the follower is not in communication range of any other robot. A complex collision resolution strategy

is employed. If nothing is seen by the centre sensor then reversing only occurs when a collision occurs, otherwise reversing is dependent on other sensor readings. Communication alters the performance of the strategy only when a target is first sensed in the centre and then nothing is sensed on the right. In this situation, if the follower is out of communication range then nothing happens, otherwise it reverses.

This program was tested to see its dependency on communication, which was achieved by first ascertaining the fitness of the program in test environments as evolved, then obtaining the fitness of the program in the same test environments with the communicating primitives replaced in turn by a constant zero. The fitnesses obtained are shown in Table 6.4. Since the GP is attempting to minimise the constraints it can be see that the performance of this controller is best when one or none of the communication receive operators is present. The reason for the bad fitness value of the initial program is not that it does not bring the robots together, it is that it brings them too close together. In fact, they touch head on and spend the majority of their time trying to break free. These results show that the performance of the controller is best when the communication operator in the reverse strategy is not used, which leaves room for further possible optimisation of the controller. Here communication is useful but its usefulness depends on where and when it is used.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 506 | 53 |
| No reverse com | 453 | 27 |
| No OnLSL com | 493 | 62 |
| All coms removed | 463 | 28 |

Table 6.4 Communication dependency fitness table for Figure 6.9.

The initial and most prevalent use of the communication was to produce straight liners with simple collision resolution strategies. Figure 6.10 shows a simple example, which employs no collision resolution at all. The performance of this controller is not altered by the removal of the communication operator.

```
system[
    main{0,-1}11[ (OnLSL (GSCom) 0.519372)]_Main
 ]_Sys
```

Figure 6.10 Indifferent application of communication.

A more effective program, which emerged around generation 40, is given in Figure 6.11. This program did not use the optimal single collision call but used two calls. The performance of this program did change with the removal of the communication operators. In this case, the outcome was opposite to the controller of Figure 6.9. The fitness is best when one or more of the communications receive operators is used. This program has evolved the optimal placing and usage of the communicated information it receives.

```
system[
    main{0,-1}11[
        (OnLSC
            (PA
                (BK (GSCom))
                (*
                    (RInv (NoMv))
                    0.41463
                )
            )
            (RInv
                (-
                    (Bk (+ (GSCom) r?))
                    0.16336
                )
            )
        )
    ]_Main
]_Sys
```

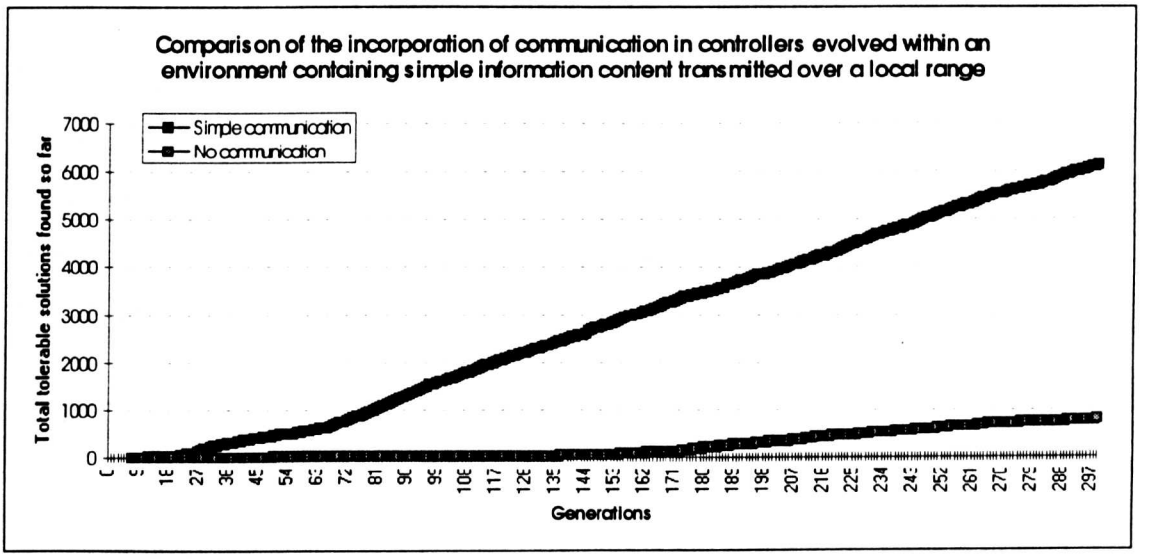
Figure 6.11 Controller exhibiting optimal placing and usage of communication.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 335 | 13 |
| No reverse com 1 | 444 | 96 |
| No reverse com 2 | 412 | 40 |
| No coms at all | 463 | 110 |

Table 6.5 Communication dependency fitness table for Figure 6.11.

Graph 6.1 shows two plots, one representing the cumulated total of tolerable controllers evolved which have the information content operator present (or ICOP) and the other the cumulated total of tolerable controllers evolved which didn't (non-ICOP). From the graph, it is obvious that the vast majority of the tolerable solutions had ICOP, indicating

that the vast majority of the controllers used communicated information in some form or other. Although the problem can be solved without the need of communication, the high numbers of solutions using communication indicate it offers an advantage over no communication. The graph also shows that the first solutions found were ICOP and there was a time lag of on average 8 generations between the first ICOP based controller and the first non-ICOP based controller being found. In addition, the steep gradient of the ICOP curve indicates an effective, prolonged use of the simple information content transmitted was quickly and easily established.




Graph 6.1 A comparison of the number of solutions produced in a simple information environment which did or did not use this locally available information.

**Displacement Communication:**

It took on average 14 generations to find the first tolerable controller and the average fitness of these controllers was [491 109]. The average time taken before the best controller emerged was 173 generations and the average fitness for these controllers was [407 41].

The majority of the initial controllers produced were only effective at close range. However, as time progressed, controllers capable of both near and far performance

slowly started to emerge. The system employed displacement communication in a similar fashion to that of simple communication. However, because of the added granularity of this form of information it resulted in the generation of more complex behaviours. The concept of action triggers was also employed here but the communication outer boundary was no longer the only trigger point. Instead, any point within the boundary could be used and these could be different for each axis.

The displacement values alone could be used to make the robot turn, which resulted in the production of controllers which were straight liners when outside the communication range, and snakers or turners when inside. Figure 6.12 shows such a controller and Table 6.6 the corresponding fitness. As is the case with many of these controllers, the performance is better when communication is removed. This shows that the direct use of these values has limited direct co-ordination powers. That is, the information communicated requires some level of processing before it can be used for co-ordination.

```
system[
    main{0,-1}12[
        (-
            (OnLSC (Gdx) 0.520486)
            (Bk 0.21534)
        )
    ]_Main
]_Sys
```

Figure 6.12 Controller misusing communicated information.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 511 | 14 |
| No Gdx | 471 | 6 |

Table 6.6 Communication dependency fitness table for Figure 6.12.

An example program is shown in Figure A.6 of Appendix A. This acts as a straight liner when out of communication range otherwise it acts as a snaker. It contains many communication receive calls. However, only two of these make any impact on performance, as can be seen from the results in Table 6.7. The controller is based on an IF statement which always evaluates to true regardless of the use of the communication

in its conditions. The only role performed in the condition section is the setting of the reversal distance. In the main true section of the controller, both Gdy and Gdx are employed. They are used as part of the turn angle calculation. Gdx is used in its inverse form, so the closer you are the larger the value used and the further away you are the smaller the value used. This indirect use of communicated information for co-ordination is better suited to the information content here.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 459 | 73 |
| No *, Gdy | 474 | 84 |
| No RInv, Gdx | 471 | 74 |
| Removed Other com | 459 | 73 |
| No com | 474 | 84 |

Table 6.7 Communication dependency fitness table for Figure A.6 appendix A.

The controller in Figure 6.13 does not employ any collision resolution strategy at all. However, it has developed a method that performs well if the robots are close to each other or facing one another. This kind of controller started to appear in small numbers around generations 20 to 30 then became quite dominant around generation 45. This is due to their effectiveness at moving close together. The controller in Figure 6.13 uses only the Gdy and applies it in two places. The first use of Gdy is as a possible parameter for the PA operator. This use implies the closer you are the more likely you are to perform the action. The second use of Gdy is as part of the turn angle calculation. As can be seen from the results in Table 6.8, the controller is highly dependent on the communication for its performance.

```
system[
    main{0,-1}12[
        (OnLSC
            (+
                (PA
                    (RInv
                        (OnLSL (NoMv) (Gdy))
                    )
                    r?
                )
                (+
                    (+
                        (Gdy)
                        (RInv  0.39519)
                    )
                    (-  0.00403 (NoMv))
                )
            )
            0.54604
        )
    ]_Main
]_Sys
```

Figure 6.13 Effective controller employing no collision resolution strategy.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 362 | 111 |
| No first Gdy | 531 | 125 |
| No second Gdy | 521 | 131 |
| No com | 517 | 133 |

Table 6.8 Communication dependency fitness table for Figure 6.13.

The high collision values in those controllers which used communication, were primarily due to head on contacts between the robots. The later programs employed incomplete reverse strategies, which were dependent on visual circumstances. In the majority of these, the communication played a scaling role in determining the distance reversed. The controllers in Figures 6.14 and 6.15 demonstrate this. By using the communicated information as the second parameter of a division call within the Bk operator, large reverse distances can be generated. The corresponding fitnesses can been seen in Table 6.9 and 6.10. In the first example, Gdx is used as part of an equation. In the second, both Gdx and Gdy are used in a simple division. This method generates the most significant reversals. The robots find it very hard to come together when the operator is executed. What the use of both these forms of information do is determine a threshold beyond which the robot cannot cross. The first method has this point close to the robots along a single axes, the second uses both axes to determine a more substantial coverage. However, with such coverage, the robots are kept far apart and the controllers

performance suffers as a result. The fitness results show this as well as indicate that this use of communicated information is beneficial.

```
system[
    main{0,-1}12[
        (OnLSC
            (-
                (*
                    (Bk
                        (Bk
                            (SH
                                (Max
                                    (/
                                        (/ x?  0.65198)
                                        (Flip (Gdx))
                                    )
                                    (/
                                        (* (NoMv) (NoMv))
                                        (OnLSR (NoMv) (Gdx))
                                    )
                                )
                            )
                        )
                    )
                    (_)
                )
                (NoMv)
            )
            0.60126
        )
    ]_Main
]_Sys
```

Figure 6.14 Controller demonstrating single axis closeness threshold.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 429 | 81 |
| No Flip Gdx | 557 | 109 |
| No OnLSR Gdx | 444 | 96 |
| No com | 573 | 122 |

Table 6.9 Communication dependency fitness table for Figure 6.14.

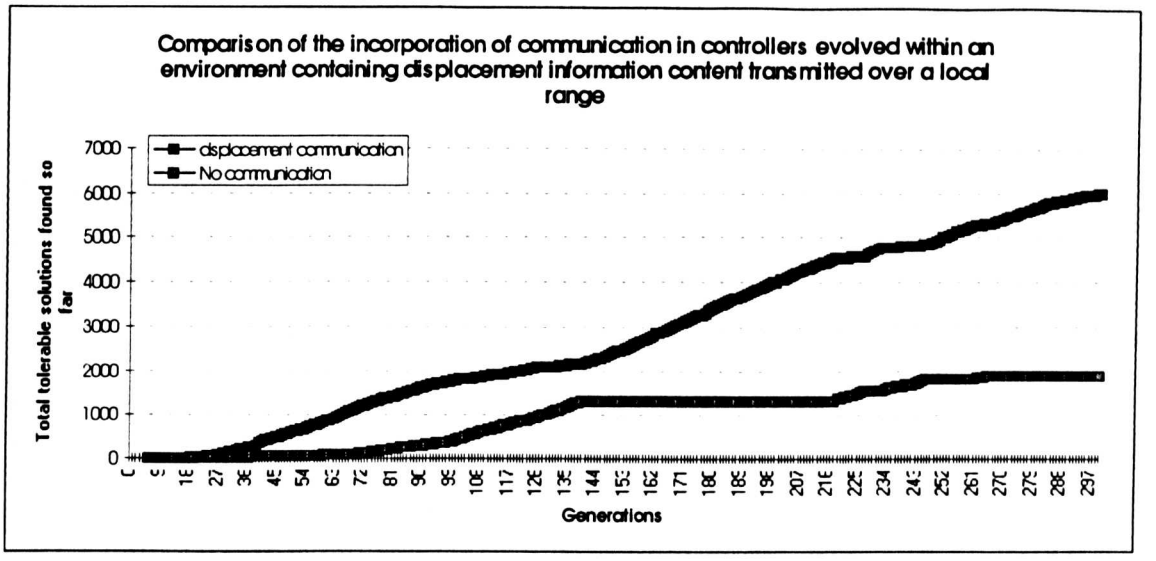
```
system[
    main{0,-1}12[
        (OnLSC
            (*
                (Gdy)
                (Bk
                    (/ (Gdx) (Gdy))
                )
            )
            0.56030
        )
    ]_Main
]_Sys
```

Figure 6.15 Controller demonstrating dual axis's closeness threshold.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 645 | 118 |
| No Gdy | 648 | 136 |
| No Gdx | 662 | 139 |
| No com | 662 | 139 |

Table 6.10 Communication dependency fitness table for Figure 6.15.

Graph 6.2 shows two plots, one representing the cumulated total of tolerable ICOP controllers evolved and the other the cumulated total of non-ICOP controllers. As in Graph 6.1 there is a time lag between the start of the production of ICOP controllers and non-ICOP controllers, with the former starting earlier. The production of ICOP controllers increases faster than non-ICOP ones. This increase was at a slower rate than in Graph 6.1. The number of non-ICOP based controllers have a larger presence in Graph 6.2 than in Graph 6.1. This can possibly be attributed to the slower increase in the number of ICOP controllers, allowing more time and space for non-ICOP solutions to become established. This slower rate of increase may be because the evolutionary process requires more time to find an effective use and role for the communicated information content. Here the communicated information can have a range of values and also that there are two communication receive operators to choose from. Although the presence of non-ICOP based controllers was greater in Graph 6.2 than in Graph 6.1 the number of ICOP based controllers still greatly outweighed them.




Graph 6.2 A comparison of the number of solutions produced in a displacement information environment which did or did not use this locally available information.

### 6.4.2.2 Discussion of results from experiment I

These experiments have shown that the evolutionary process is capable of evolving programs which can take advantage of the presence of communicated information (a summary of the performance of the controllers can be seen in Table 6.11). It also showed that the misuse of communication can produce controllers which underperform. However, given sufficient time the evolutionary process can evolve optimal usage and positioning of communication receive operators. The evolutionary process is also able to take advantage of granularity when it is present in the information. This was exhibited in its use of Gdx and Gdy operators, where it was able to more precisely fine tune and target behaviours proportional to the distance of a communicating robot. This ability to target behaviours more precisely did not produce any significant performance benefit. Optimal, or near optimal use of communication consistently resulted in more robust controller among the high performers in many scenarios. Further, the use of communication is effectively a behaviour trigger in both cases. To establish if the use of communication is limited to behavioural triggers or whether this is a factor of communication range and whether the complexity of the information transmitted offers any significant performance improvement, the testing was changed such that the

information was transmitted globally instead of locally. These experiments are presented in section 6.4.3.

| CHAPTER 6 FIGURE REFERENCE | SUMMARY OF CONTROLLER | FITNESS USING COMMUNICA-TION | | FITNESS USING NO COMMUNICA-TION | |
|---|---|---|---|---|---|
| | | constraint 1 | constraint 2 | constraint 1 | constraint 2 |
| 9 | Performs better with less or no communication *(GSCom).* | 506 | 53 | 463 | 28 |
| 11 | Effective use of communication as well as collision resolution strategy *(GSCom).* | 335 | 13 | 463 | 110 |
| 12 | Misuse of communicated information *(Gdx; Gdy).* | 511 | 14 | 471 | 6 |
| 13 | Effective use of the communicated information but no collision resolution strategy *(Gdx; Gdy).* | 362 | 111 | 517 | 133 |
| 14 | Communication only employed in collision resolution strategy *(Gdx; Gdy).* | 429 | 81 | 573 | 122 |
| 15 | Employing communicated information in both co-ordination and collision resolution *(Gdx; Gdy).* | 645 | 118 | 662 | 139 |

Table 6.11 Summary table of performance of the evolved controllers presented in experiment I.

### 6.4.3 Experiment II

*Aim:* To determine if the role played by communicated information in the presence of visual guidance operators changes if the communication range is no longer restricted.

The range of both simple and displacement communication is extended to cover the whole environment. This global communication is continuously and involuntarily produced by the robots.

#### 6.4.3.1 Results of experiment II

**Simple communication:**

No controllers emerged which used the communicated information in any meaningful way. The information content offered no additional benefit to the evolutionary system, its role here effectively being a constant value.

**Displacement Communication:**

The average time taken to find the first tolerable controller was 3 generations. The average fitness of these controllers was [485 137]. The optimisation of the controllers continued for on average 175 generations and the average fitness for them was [359 13].

The information content offered benefit to the evolutionary system, which resulted in the production of communication dependent controllers. The vast majority of the solutions produced used communication. The trigger nature exhibited in the local case had no significant presence here, in those cases where it did exist it was always explicitly stated and much subtler. The form this explicitness took can be seen in Figure 6.16 and Figure 6.17.

The controller in Figure 6.16 uses the trigger to determine whether to use Gdx as a factor in the turn angle calculation. This occurs in an IF statement in which Gdy is compared with 0.17816. The trigger translates as follows: *if a communicating robot is less than 21.3792 (0.17816) units below me then Gdx should not be a factor in my co-ordination strategy otherwise it should.* The results shown in Table 6.12 indicate that the presence of this trigger is highly beneficial to the performance of the controller. Further, the controllers use of communication in general is also of benefit to it.

```
system[
    main{0,-1}12[
        (+
            (IFGE
                 0.28718
                (OnLSR
                    (*
                        (OnLSC
                            (OnLSL (OnLSC (Gdx) (Gdy)) (Gdx))
                            (Bk (Gdy))
                         )
                         0.78991
                     )
                    (_)
                )
                  THEN
                   (Bk
                        (OnLSR
                            (*
                                (Bk
                                    (+
                                        (OnLSR  0.22635  0.69735)
                                        (NoMv)
                                    )
                                 )
                                 (Min (Gdx) (Gdx))
                             )
                             0.87332
                         )
                     )
                  ELSE
                   (-
                        (OnLSC
                            (-
                                (IFGE  0.17816 (Gdy)
                                   THEN ( )
                                   ELSE (Gdx)
                                )
                                (Flip (_))
                             )
                            (+
                                (Gdx)
                                (/ r? (Gdx))
                             )
                         )
                        (Gdx)
                    )
            )
            0.64021
        )
    ]_Main
]_Sys
```

Figure 6.16 Controller featuring the use of explicit communication dependent behaviour trigger.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 348 | 4 |
| No trigger | 469 | 44 |
| No com | 461 | 36 |

Table 6.12 Communication dependency fitness table for Figure 6.16.

The trigger used in Figure 6.17 is achieved with the Min statement. It translates as follows: *when a communicating robot is less than 20.0012 units to the right of me then Gdx should be used for direct co-ordination otherwise I should turn through 36.02° (20.0012) anticlockwise.* The results shown in Table 6.13 indicate that the controllers use of communication is detrimental as well as non-optimal for its structure. The improvement obtainable by using just one of the communication cases

```
system[
    main{0,-1}12[
        (-
            (OnLSC
                (Min
                    (Gdx)
                    0.20012
                )
                0.56294
            )
            (Bk
                (/
                    (NoMv)
                    (+ (_) (Gdy))
                )
            )
        )
    ]_Main
]_Sys
```

Figure 6.17 Controller exhibiting subtle explicit communication dependent behaviour trigger.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 456 | 8 |
| No Gdx | 391 | 39 |
| No Gdy | 410 | 29 |
| No com | 423 | 69 |

Table 6.13 Communication dependency fitness table for Figure 6.17.

was exploited by the evolutionary process in subsequent generations and many such variants later emerged. The optimising of this particular controller did not stop here. About 10 generations after the emergence of the controller an improved version

appeared (see Figure 6.18), in which the use of communication was not detrimental as the result in Table 6.14 shows, although its use was still not optimal. The optimisation of the controller continued, and some 40 generations later, an improved version emerged (see Figure 6.19). From Table 6.15 it can be seen that the controller is now optimal in its use of communication as well as benefiting from its exploitation. This ability of the evolutionary process to optimise controllers for the overall use of communication or for its general presence lead to the development of controllers which either fell into one or both of these categories. The majority of the controllers tended to fall in to the latter one. To accomplish both these strategies, the evolutionary process applied one or both of the following strategies, either remove the usage of communication, which is detrimental, or fine tune parameters or structures within the controller to complement the use of communication.

```
system[
    main(0,-1)12[
        (-
            (OnLSC
                (Min
                    (Gdx)
                    0.15016
                )
                0.56294
            )
            (Bk
                (/
                    (NoMv)
                    (+ (_) (Gdy))
                )
            )
        )
    ]_Main
]_Sys
```

Figure 6.18 Improved version of controller in Figure 6.17.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 419 | 16 |
| No Gdx | 391 | 39 |
| No Gdy | 493 | 61 |
| No com | 423 | 69 |

Table 6.14 Communication dependency fitness table for Figure 6.18.

```
system[
    main(0,-1)12[
        (-
            (OnLSC
                (Min
                    (Gdx)
                    0.09901
                )
                0.56294
            )
            (Bk
                (/
                    (NoMv)
                    (+ (_) (Gdy))
                )
            )
        )
    ]_Main
]_Sys
```

Figure 6.19 Most efficient version of the controller in Figure 6.17 evolved.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 347 | 7 |
| No Gdx | 391 | 39 |
| No Gdy | 354 | 18 |
| No com | 423 | 69 |

Table 6.15 Communication dependency fitness table for Figure 6.19.

The controllers in Figures 6.20 and 6.21 do not use triggers, their respective fitness can be seen in Tables 6.16 to 6.19. These snaker-based controllers were also capable of taking full advantage of the presence of communicated information to improve their performance. Both Gdx and Gdy play roles within the controllers turn angle calculation as well as their reversal strategy.

```
system[
    main{0,-1}12[
        (OnLSC
            (OnLSR
                (-
                    (+ (PA r? (Gdx)) (Bk (Gdx)))
                    (RInv
                        (Max
                            (OnLSC
                                (-
                                    (Max
                                        (Max (+ r? (Gdy)) (Gdy))
                                        (Bk (Max  0.93504 (Gdy)))
                                    )
                                    (Bk (_))
                                )
                                (IFGE
                                    (*  0.58614  0.34831)
                                    0.23362
                                      THEN (Gdx)
                                      ELSE
                                        (-
                                            (OnLSL (Gdy) (Gdy))
                                            0.31679
                                        )
                                )
                            )
                            r?
                        )
                    )
                )
                (Bk (OnLSC (NoMv) (NoMv)))
            )
            (Max (Bk (OnLSC (Max  0.84032  0.82354) r?)) 0.14450)
        )
    ]_Main
]_Sys
```

Figure 6.20 Snaker based controller fully utilising communicated information.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 441 | 45 |
| No com | 444 | 39 |

Table 6.16 Communication dependency fitness table for Figure 6.20.

```
system[
    main{0,-1}12[
        (OnLSC
            (IFGE
                (NoMv)
                (Bk
                    (IFGE
                        (_)
                        (OnLSC
                            (/
                                (OnLSL r? (Gdx))
                                (OnLSC (NoMv) r?)
                            )
                            (* (* (NoMv) r?) (Max (Gdy) (Gdy)))
                        )
                        THEN 0.14761
                        ELSE (NoMv)
                    )
                )
                THEN (OnLSC (NoMv) (Gdx))
                ELSE (Gdx)
            )
            (Max
                (Bk (+ 0.12367 (OnLSR 0.81169 (Gdy))))
                0.14113
            )
        )
    ]_Main
]_Sys
```

Figure 6.21 Another controller which fully utilises all the available communicated information.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 412 | 71 |
| No com | 701 | 94 |

Table 6.17 Communication dependency fitness table for Figure 6.21.

The majority of the controllers produced were straight liners, but instead of setting the turn angle to zero to make them head in a straight line they used the length of the program. This approach to straight liners was widely used in both the global and local case. In all these cases, both the role and the use of communication was unclear. Examples of its use for collision avoidance are shown in Figures A.7 and A.8 of Appendix A. The results for these controllers given in Tables 6.18 and 6.19 show that although the exact role and usage of communication maybe unclear, its overall use is still of significant benefit to the controllers.

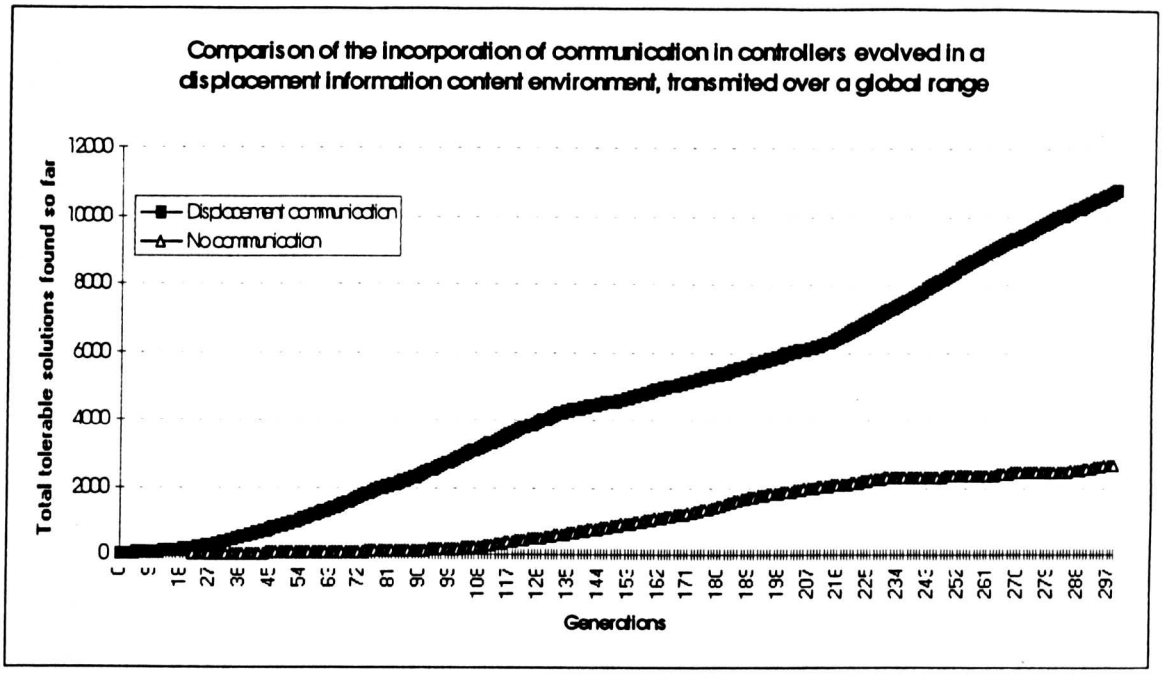
| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 435 | 38 |
| No com | 551 | 28 |

Table 6.18 Communication dependency fitness table for Figure A.7 appendix A.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 391 | 20 |
| No com | 395 | 34 |

Table 6.19 Communication dependency fitness table for Figure A.8 appendix A.

Graph 6.3 shows two curves, one representing the cumulated total of tolerable ICOP controllers evolved and the other the cumulated total of non-ICOP controllers. The time lag between the start of the production of ICOP controllers and non-ICOP controllers is still present here, its average duration is increased to 22 generations. Production of the ICOP controllers always began first and increased at faster rate than the non-ICOP ones. This rate of increase was also faster than that achieved when a local communication range is used regardless of information content (as shown in Graph 6.1 and Graph 6.2). The number of non-ICOP based controllers have a similar final presence to that of Graph 6.2 but their rate of increase is initially a lot slower. This can possibly be attributed to the very fast increase in the number of ICOP controllers, allowing less time and space for non-ICOP solutions to become established. The early use of communication in the controllers as well as the large total number tolerable controllers produced, indicates its benefits and possible use are much easier to establish when the information content is globally available.




Graph 6.3 A comparison of the number of solutions produced in a displacement information environment which did or did not use this globally available information.

### 6.4.3.2 Discussion of results from experiment II

From these experiments, it can be seen that the content of the information being received is important. These results, in conjunction with those from section 6.4.2, indicate that this importance is dependent on the range of communication. The use of simple communication in this experiment offered no significant benefit, since the global use rendered it a constant, resulting in no meaningful information being conveyed. Its good performance in *experiment I* came primarily from the implicit trigger nature offered by the communication range. As for the use of displacement information, its complex nature allowed its usefulness to be independent of communication range. The results indicate that in order for communication to be useful in a global sense, it should have or convey some form of structure or granularity (i.e. range of values).

The trigger nature of communication, which was so predominant when local communication is used, was not very evident here. The vast majority of the effective controllers evolved did not exhibit any trigger-based behaviour. However, those that

did, had to state it explicitly. The displacement-based controllers produced were better than their local counterparts with respect to their overall performance. Also the evolution of effective solutions was a lot easier. The behaviour of these controllers was consistent and they exhibited greater awareness of each other. This greater awareness of each other is highlighted by the better performance of the controllers in test cases where the robots were far apart. A summary of the performance of the controllers reported in this section can be seen in Table 6.20.

The major component of the fitness of the controller produced in this and the previous experiments can be attributed to the use of visual functions. At no time did a controller emerge which depended solely on communication. Therefore, the use of communication in these experiments was a means to fill in for the shortcomings or lack of resolution of the visual functions. This leads to the possible conclusion that communication is not required for the task in the presence of direct visual ability. It could just be that these controllers take longer, or are harder to evolve, or that the wrong kind of information content is being utilised. In a bid to establish which of these cases, if any is true, the communicated information was altered and the direct use of visual information removed. This also allows for a comparison of performance of the ability to utilise communicated information under weak (as is the case in this and the previous experiment) and strong evolutionary pressures (as is the case in the next experiment). Section 6.4.4 reports on the details and the results of the experiments to test these points.

| CHAPTER 6 FIGURE REFERENCE | SUMMARY OF CONTROLLER | FITNESS USING COMMUNICA-TION | | FITNESS USING NO COMMUNICA-TION | |
|---|---|---|---|---|---|
| | | constraint 1 | constraint 2 | constraint 1 | constraint 2 |
| 16 | Utilises explicit communication based trigger *(Gdx; Gdy).* | 348 | 4 | 461 | 36 |
| 17 | Initial occurrence of controller exhibiting unoptimise usage of communicated information *(Gdx; Gdy).* | 456 | 8 | 423 | 69 |
| 18 | Improved occurrence of Figure 6.17 giving better performance *(Gdx; Gdy).* | 419 | 16 | 423 | 69 |
| 19 | Final and fully optimised occurrence of Figure 6.17 *(Gdx; Gdy).* | 347 | 7 | 423 | 69 |
| 20 | Fully utilisation of communicated information *(Gdx; Gdy).* | 441 | 45 | 444 | 39 |
| 21 | Fully utilisation of communicated information *(Gdx; Gdy).* | 412 | 71 | 701 | 94 |

Table 6.20 Summary table of performance of the evolved controllers presented in experiment II.

### 6.4.4 Experiment III

*Aim:* To determine the role communicated information will play in the absence of any other form of global positional information between robots and the way information content affects its potential use.

In this experiment no visual operators are available to the evolution process. The task must be completed using the communicated information content only, which can be

considered as placing a stronger evolutionary pressure on the use of communication. Global communication is used predominantly here. Two additional forms of information content are introduced and are used in conjunction with the existing displacement communication presented in section 6.4.1. These new forms of communicated information content are heading based displacement and visual sighted notification.

**Heading based displacement x, y**

This is similar to the existing displacement case except that the receiving robot uses its own local axis, which is dependent on its own heading, to generate the displacements. The two operators, which accomplish this, are GRdx and GRdy. This form of communication was used and defined by Reynolds [124].

**Visual sighted notification:**

This requires the transmission of a single value, which indicates whether the transmitting robot can see the receiving robot in a particular visual sector. The values 0.0 (not seen), 0.125 (seen on left), 0.25 (seen in centre) and 0.375 (seen on right) are used. The GVI operator is used for this.

### 6.4.4.1 Results of experiment III

**Displacement communication:**

The evolutionary process found it very difficult to produce adequate controllers which just utilised the communicated information. However, as poor as the controllers were, the dependency on the communicated information received for performance was still visible. Figures A.9 to A.11 of appendix A, show three such controllers, which, when their communication receive operators are removed, experience a resulting drop in fitness ranging from 3 to 113 percent (see Tables 6.21 to 6.23).

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 684 | 32 |
| No com | 709 | 40 |

Table 6.21 Communication dependency fitness table for Figure A.9 appendix A.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 686 | 35 |
| No com | 1201 | 130 |

Table 6.22 Communication dependency fitness table for Figure A.10 appendix A.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 534 | 12 |
| No com | 1140 | 117 |

Table 6.23 Communication dependency fitness table for Figure A.11 appendix A.

The lack of success of these controllers prompted some tests using local communication instead of global. In these tests, the extreme difficulties apparent in the global case were not encountered. Although a large number of controllers did emerge which only performed well when the robots were in close proximity to each other, a reasonable numbers of controllers with all round good performance did start to be produced later on. It took on average 15 generations to find the first tolerable solution. The average fitness of these controllers was [498 115]. The optimisation of the controllers continued for on average 110 generations. The average fitness for the best controller was [394 35].

The strategies of these controllers took advantage of the implicit trigger-based nature offered by local communication. The basic strategy of these controllers was to turn through a circle of large radius until the robot comes into the communication range of another, then employ the communicated information for indirect co-ordination. Again, the controllers produced were all dependent on the communicated information received for their performance. Figures 6.22 to 6.24 show three such controllers and Tables 6.24 to 6.26 show their respective fitness.

```
system[
    main{0,-1}7[
        (+
            (Null (+ (_)  0.59751))
            (+
                (Gdy)
                (Bk
                    (Max
                        (Gdx)
                        (-
                            (/
                                (IFGE r? (NoMv)
                                   THEN (Gdy)
                                   ELSE r?
                                )
                                (Flip  0.65309)
                            )
                            (*
                                (IFGE
                                     0.17352
                                    (*
                                        (Min
                                             0.81509
                                            (IFGE (NoMv) r?
                                               THEN (Gdx)
                                               ELSE (NoMv)
                                            )
                                        )
                                        (+ (Gdx) (_))
                                    )
                                   THEN
                                    (Null (Min (Bk (NoMv)) (Gdx)))
                                   ELSE
                                    (+
                                        (* 0.49942 (Flip (NoMv)))
                                        (NoMv)
                                    )
                                )
                                (NoMv)
                            )
                        )
                    )
                )
            )
        )
    ]_Main
]_Sys
```

Figure 6.22 Effective controller produced using displacement communication over a localised range.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 482 | 11 |
| No com | 687 | 17 |

Table 6.24 Communication dependency fitness table for Figure 6.22.

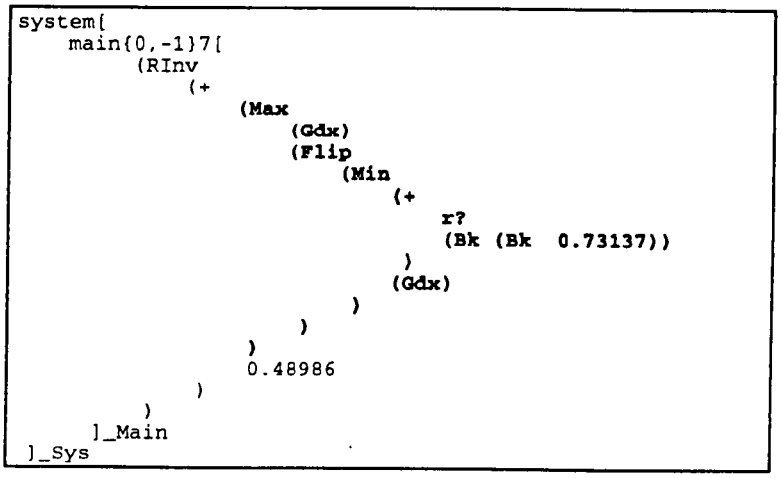



Figure 6.23 Localised communication using Gdx as part of co-ordination calculations.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 606 | 26 |
| No com | 905 | 55 |

Table 6.25 Communication dependency fitness table for Figure 6.23.

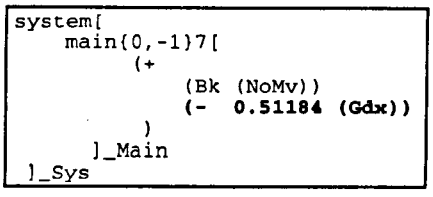


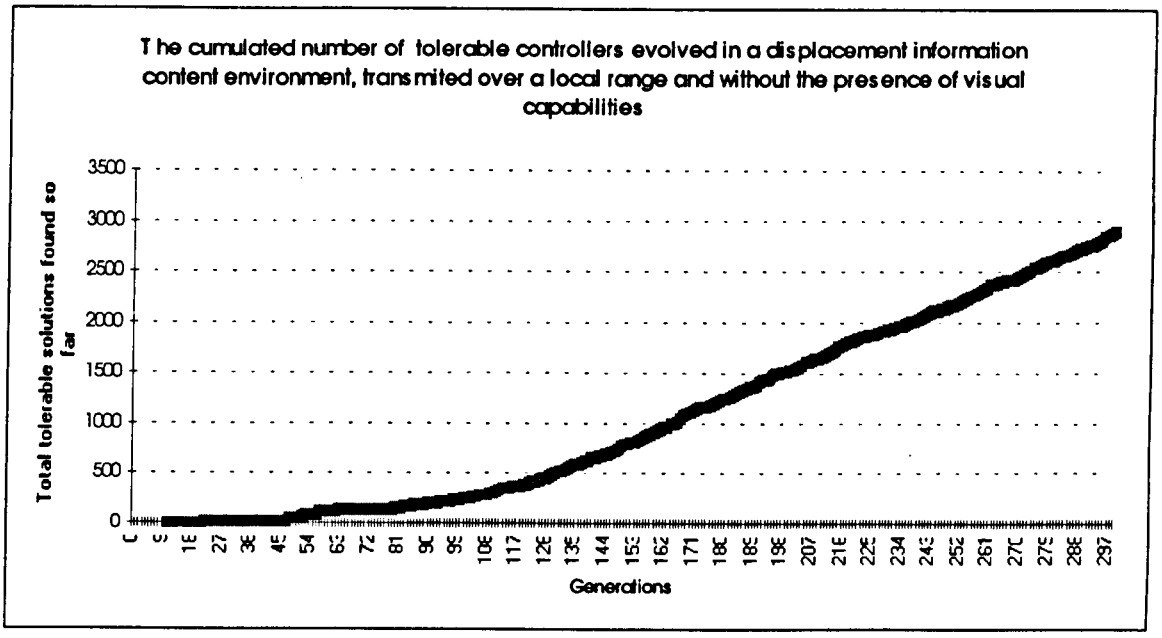

Figure 6.24 Simple controller exhibiting general behaviour of solutions using localised communication.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 592 | 20 |
| No com | 704 | 17 |

Table 6.26 Communication dependency fitness table for Figure 6.24.

Graph 6.4 shows the cumulated total number of tolerable ICOP solutions found. No non-ICOP tolerable solutions were produced since the only way for the robots to be aware of each other was via communication. This use of strong evolutionary pressure towards the use of communication actually resulted in a considerable reduction of the final total number of tolerable solutions produced. This can be seen by comparing Graph

6.4 and Graph 6.2. A similar difference is also visible between Graph 6.4 and Graph 6.3, although the communication ranges are different. From Graph 6.4 it can be seen that the production of tolerable controllers is initially very slow only really starting to increase relatively rapidly after about 130 generations. This and the low final tally can be attributed to the difficulty in generating solutions which use the communicated information content in an effective fashion.




Graph 6.4 Cumulated total of tolerable solutions produced when the use of communicated information is the only method available for the robots to be aware of each other and when the information content contains displacement information.

**Heading Displacement:**

The average time taken to find the first tolerable controller was 2 generations and the average fitness of these controllers was [445 67]. It took on average a further 65 generations before the best controller was produced. The average fitness for these controllers was [341 17].

The communication of heading-based displacement information was very well suited to the task being undertaken. The performance of the solutions produced was comparable

to that of the visual operator cases. Their performance was significantly more closely linked to the presence of the communicated information. Effective solutions, which tended to be quite short, were easily found starting from the initial population. The initial controllers found did not employ any collision resolution strategy though. This persisted for a while, owing to the effectiveness of the communication in the controllers. Examples of these controllers can be seen in Figures 6.25 and 6.26 and their respective fitness in Tables 6.27 and 6.28. In both these cases the communicated information was used directly to co-ordinate the robot. The controller in Figure 6.25 utilises both the available information types in a probabilistic fashion.

```
system[
    main{0,-1}8[
        (/
            (GRdx)
            (RInv
                (PA
                    (NoMv)
                    (+
                        (- (NoMv) (_))
                        (Max (NoMv) (_))
                    )
                )
            )
        )
    ]_Main
]_Sys
```

Figure 6.25 Initial controller utilising no collision resolution strategy and direct co-ordination with communicated information.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 626 | 138 |
| No com | 1268 | 160 |

Table 6.27 Communication dependency fitness table for Figure 6.25.

```
system[
    main{0,-1}8[
        (+
            (PA (GRdx) (GRdy))
            (GRdx)
        )
    ]_Main
]_Sys
```

Figure 6.26 Controller exhibiting probabilistic communication based trigger.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 540 | 147 |
| No com | 1268 | 160 |

Table 6.28 Communication dependency fitness table for Figure 6.26.

The use of collision resolution started to emerge around generation 25 Figures 6.27 and 6.28 show two such controllers whose associated fitness can be seen in Tables 6.29 and 6.30 respectively. The direct use of communicated information to co-ordinate the robots is still employed along with many other indirect usages. There was an abundance of solutions that used only GRdx, which became more apparent as time went on. This can be attributed to the fast and often erratic snaking like behaviour the use of this communication offered. This was in contrast to the more curved and sedate paths produced with GRdy.

```
system[
    main{0,-1}8[
        (+
            (/
                (NoMv)
                (/
                    (Bk (NoMv))
                     0.07965
                )
             )
            (GRdx)
        )
    ]_Main
]_Sys
```

Figure 6.27 Controller employing collision resolution strategy and direct co-ordination using communicated information.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 384 | 58 |
| No com | 1217 | 109 |

Table 6.29 Communication dependency fitness table for Figure 6.27.

```
system[
    main{0,-1}8[
        (*
            (GRdy)
            (Max
                (Bk r?)
                (RInv  0.36169)
            )
        )
    ]_Main
]_Sys
```

Figure 6.28 Controller using GRdy for co-ordination.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 465 | 16 |
| No com | 1218 | 116 |

Table 6.30 Communication dependency fitness table for Figure 6.28.

The controller shown in Figure 6.29 demonstrates that the exact value of the communicated heading displacement information is not essential to the co-ordination of the robot. In this controller the RInv operator is applied to the communicated information, which converts large angles to small angles and small angles to large ones. The fitness of this controller can be seen in Table 6.31.

```
system[
    main{0,-1}8[
        (+
            (Bk
                (Max
                    (Max (GRdx) (NoMv))
                    (NoMv)
                )
            )
            (RInv (GRdx))
        )
    ]_Main
]_Sys
```

Figure 6.29 Controller illustrating that the precise contents of information transmitted is not really relevant.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 448 | 91 |
| No com | 1211 | 104 |

Table 6.31 Communication dependency fitness table for Figure 6.29.

The majority of the solutions, which employed communicated information as part of their collision resolution strategy, used GRdy as the main parameter. An example of

such a controller is given in Figure 6.30 the corresponding fitness can be seen in Table 6.32. Few controllers evolved which used both GRdx and GRdy as part of their turn angle calculation. Examples of such controllers are given in Figures 6.31 and 6.32 and their respective fitness in Tables 6.33 and 6.34.

```
system[
    main{0,-1}8[
        (Max
            (GRdx)
            (Bk
                (/
                    (Max (GRdy) (NoMv))
                    r?
                )
            )
        )
    ]_Main
]_Sys
```

Figure 6.30 Controller demonstrating the main role of GRdy as part of reversal strategy.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 359 | 25 |
| No com | 1132 | 41 |

Table 6.32 Communication dependency fitness table for Figure 6.30.

```
system[
    main{0,-1}8[
        (/
            (/
                (-
                    (Max (GRdy)  0.89448)
                    (NoMv)
                )
                (Bk
                    (+
                        (PA r?  0.73187)
                        (NoMv)
                    )
                )
            )
            (GRdx)
        )
    ]_Main
]_Sys
```

Figure 6.31 Controller exhibiting use of all communicated information for co-ordination.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 401 | 53 |
| No com | 934 | 56 |

Table 6.33 Communication dependency fitness table for Figure 6.31.

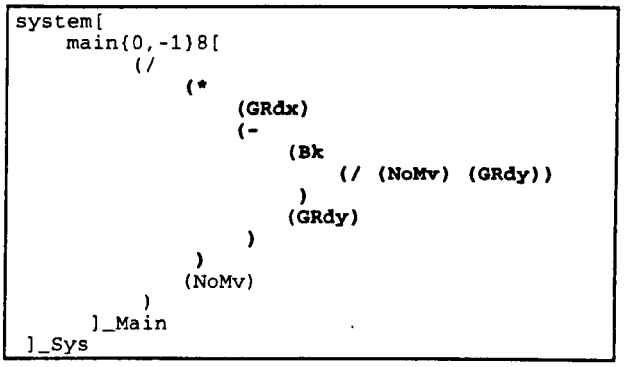
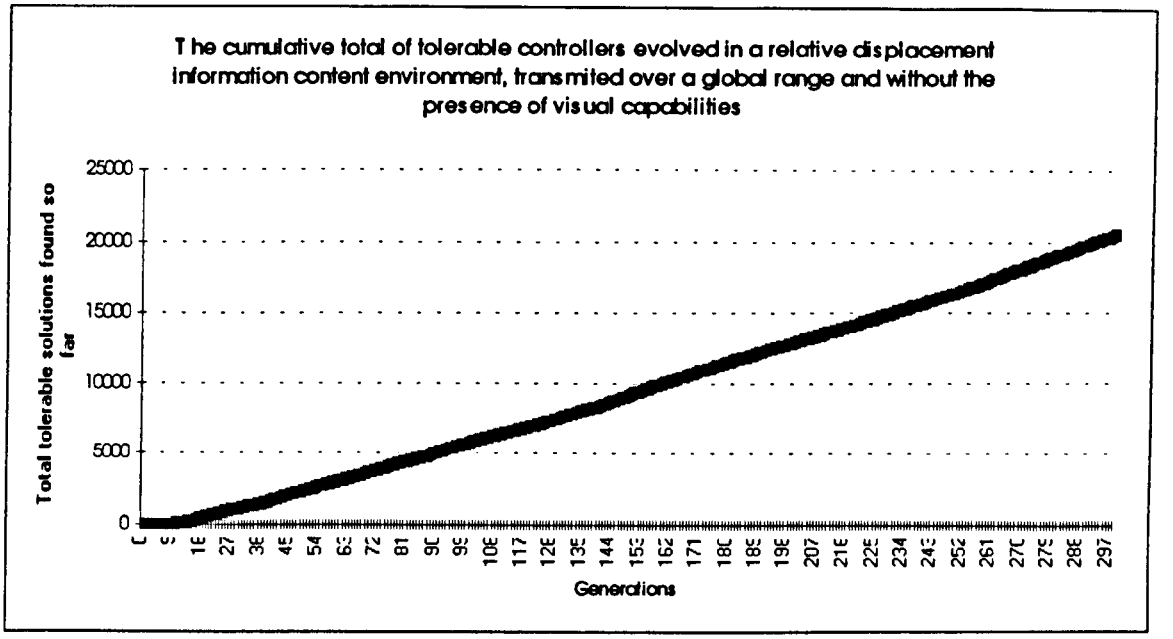



Figure 6.32 Controller showing the use of all communicated information in both co-ordination and reversal strategy.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 582 | 13 |
| No com | 1217 | 110 |

Table 6.34 Communication dependency fitness table for Figure 6.32.

Graph 6.5 shows the cumulated total number of tolerable ICOP solutions found. As in Graph 6.4 no non-ICOP tolerable solutions were produced since the only way for the robots to be aware of each other was via communication. Under this strong evolutionary pressure towards the use of communication, the final total number of tolerable solutions produced was the largest of all the experiments. Using this information content the production of controllers starts early and increases very quickly, this rapid increase is sustained for the duration of the run. This early start and the prolonged rapid increase in the number of tolerable solutions can be attributed to the suitability of the information content for the task and maybe in part to the lack of any other competing equally creditable avenue.




Graph 6.5 Cumulated total of tolerable solutions produced when the use of communicated information is the only method available for the robots to be aware of each other and when the information content contains relative displacement information.

**Sighted notification:**

The evolutionary process found it extremely hard to produce any good controllers using this communicated information. The presence of communication in the controller did offer some advantage. Figure 6.33 shows a controller which exhibits the best performance achieved using communication and Table 6.35 shows its fitness.

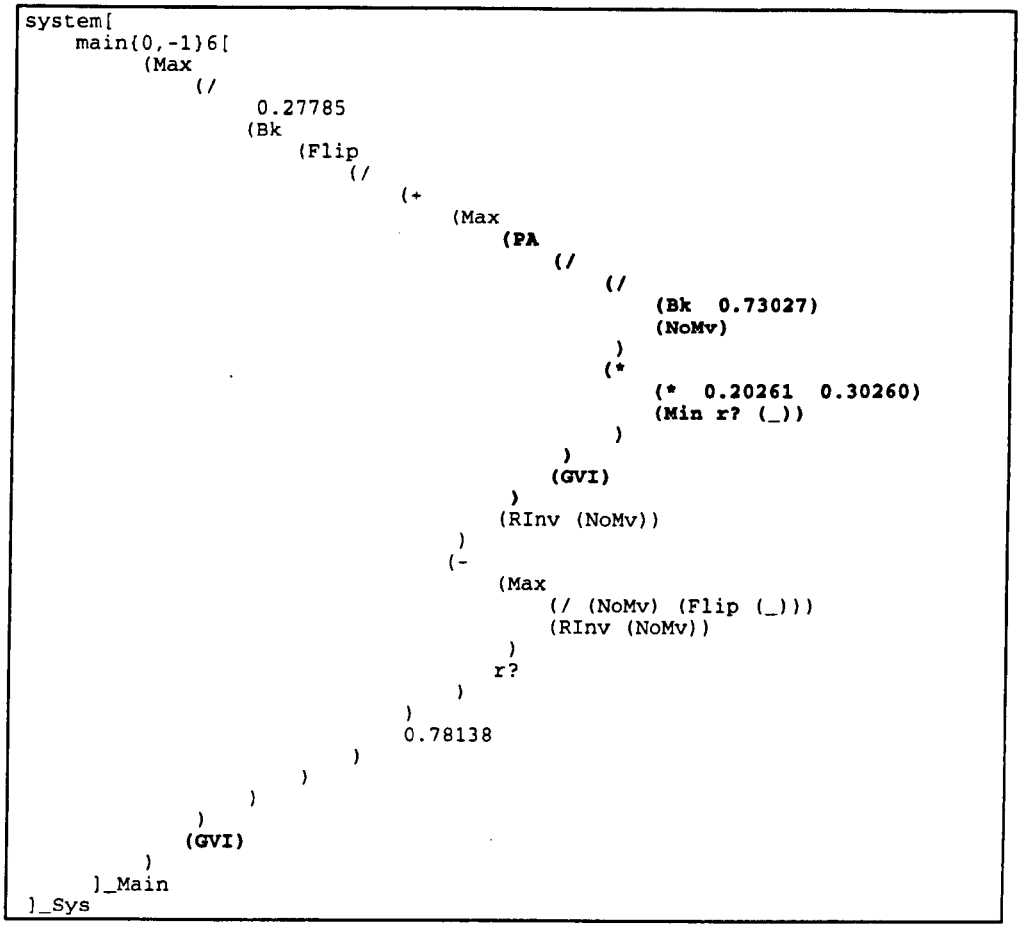



Figure 6.33 Best controller using communicated visual information.

| | Constraint 1 | Constraint 2 |
|---|---|---|
| Initial | 787 | 14 |
| No com | 798 | 24 |

Table 6.35 Communication dependency fitness table for Figure 6.33.

### 6.4.4.2 Discussion of results from experiment III

The results presented here have shown that controllers can be evolved which rely solely on communicated information as their only guide to co-ordination in conditions where communication is the only available avenue to solve the problem (a summary of the performance of the controllers reported in these experiments can be seen in Table 6.36). Although there is a stronger evolutionary pressure towards use of communication, the results from these experiments show that the problem is not necessarily made any easier, in fact in some cases it is actually made much more difficult. This is because a greater

emphasis is placed on the appropriateness of the information content for the problem. So any deficiencies or inappropriateness in the information content can not be compensated for by producing controllers which collaborate with another solution avenue. Although all the controllers produced were dependent on the presence of communicated information for their level of performance, the degree of performance they achieved was heavily influenced by the information content and in some cases the transmission range. It can also be seen that although the use of visual operators is highly beneficial the communication of such information is not. This can probably be attributed to the inability to take actions, which are relative to a robot, a facility offered by the visual operators, as well as relative displacement communication. Without this ability, the evolutionary process found it very difficult to produce consistently effective strategies. In the case of displacement communication, changing the communication range made it easier to find such strategies.

The slow evolution time until the first expectable controllers are found confirms to some degree the assertion made earlier in section 6.4.3 that the presence of visual operators hampers the development of controllers solely based on communication, by crowding them out of the population.

| CHAPTER 6 FIGURE REFERENCE | SUMMARY OF CONTROLLER | FITNESS USING COMMUNICA-TION | | FITNESS USING NO COMMUNICA-TION | |
|---|---|---|---|---|---|
| | | constraint 1 | constraint 2 | constraint 1 | constraint 2 |
| 22 | Good overall performance but dependent on communication range as well as communication *(Gdx; Gdy -local-).* | 482 | 11 | 687 | 17 |
| 23 | Same as Figure 6.22 *(Gdx; Gdy - local-).* | 606 | 26 | 905 | 55 |
| 24 | Same as Figure 6.22 *(Gdx; Gdy - local-).* | 592 | 20 | 704 | 17 |
| 25 | No collision resolution strategy *(GRdx; GRdy).* | 626 | 138 | 1268 | 160 |
| 26 | Uses probabilistic communication based trigger and no collision resolution *(GRdx; GRdy).* | 540 | 147 | 1268 | 160 |
| 27 | Uses communicated information for direct co-ordination *(GRdx; GRdy).* | 384 | 58 | 1217 | 109 |
| 28 | Uses only GRdy for co-ordination *(GRdx; GRdy).* | 465 | 16 | 1218 | 116 |
| 29 | Exact values for communicated information not required for co-ordination *(GRdx; GRdy).* | 448 | 91 | 1211 | 104 |
| 30 | Main role of GRdy as part of collision resolution strategy *(GRdx; GRdy).* | 359 | 25 | 1132 | 41 |
| 31 | Co-ordinates using | 401 | 53 | 934 | 56 |

| CHAPTER 6 FIGURE REFERENCE | SUMMARY OF CONTROLLER | FITNESS USING COMMUNICA-TION | | FITNESS USING NO COMMUNICA-TION | |
|---|---|---|---|---|---|
| | | constraint 1 | constraint 2 | constraint 1 | constraint 2 |
| | both forms of communicated information *(GRdx; GRdy).* | | | | |
| 32 | Communication dependent reversal strategy *(GRdx; GRdy).* | 582 | 13 | 1217 | 110 |
| 33 | The best controller using communicated visual information *(GVI).* | 787 | 14 | 798 | 24 |

Table 6.36 Summary table of performance of the evolved controllers presented in experiment III.

## 6.5 SUMMARY

The work presented in this chapter illustrates that the evolutionary process can benefit from the presence of communicated information. It can apply this information and optimise the structure as well as the parameters within evolved controllers to exploit the benefit offered by a particular form of communicated information over a specific transmission range. It can also utilise this information to compensate for shortcomings in visual operators or as the sole method for making robots aware of each other for co-ordination purposes. Different forms of information offer the evolutionary process alternative ways to use them. Some information is best suited for reversal strategies where as others are more suited for co-ordination. Although the transmission is continuous, the evolutionary process is capable of producing controllers which effectively ignore the communicated information until certain circumstances arise. This it does by using explicit triggers.

It has been shown that under weak evolutionary pressures, towards the use of communicated information, that less emphasis is placed on the appropriateness of information content communicated in order complete the task. However, under conditions of stronger evolutionary pressures, a much greater emphasis is placed on the appropriateness of information content to the task. Due to this increased emphasis on information content the task to be solved does not necessarily become easier when a stronger evolutionary pressure is present. What it does show is that the more meaningful the information communicated is to the task, the faster the evolutionary process takes up its use and the greater the evolved controllers dependency on the information will be.

So, in summary, the key findings are that firstly care most be taken over the choice of communication range as well as information content as these impact greatly on the evolutionary process ability to find an application for and make effective use of communicated information. Second, and finally, the evolutionary process is capable of making use of communicated information in the presence of both weak and strong evolutionary pressures on its use. This can be seen from the results of sections 6.4.2 and 6.4.3 (weak pressure) where a use for communicated information is found in the presence of alternate modes of solution and in section 6.4.4 (strong pressure) when communication is the only means of accessing internal information essential to the task.

The interaction with the real world produces a myriad of streams of information and associated responses, the ability to handle and prioritise these streams is a major pre-requisite for any robot or robotic system. This work has shown that the genetic programming approach to the development of control systems offers a flexible and effective method for fusing and prioritising multiple streams of information generated from interactions with an environment.

The broader implications of these results as they apply to the field of robotics research in general are now discussed. The ability to find a balance between competing tasks or processes within a system is an important property, since typically any task can be

decomposed into a number of sub-task each of which contributes to the problem. This touches on areas of sensor fusion, task scheduling and task decomposition.

The ability to combine and utilise visual information and communicated information is an example of sensor fusion, and the effective prioritising and integration of competing sub-tasks are examples of task scheduling and decomposition. By either defining the competing tasks used here as distinct entities, which operate independently of each other, or integrating them into a single indivisible unit, are examples of the task decomposition ability of this approach.

The handling of competing sub-tasks or layers/behaviours was one of the problems associated with the subsumption architecture approach [3]. In addition, the ability to handle and identify varying levels of task decomposition is a drawback of traditional AI approaches to robotic systems and to some extent the subsumption architecture, but one which is handled by this approach.

In summary, these results can be translated as showing genetic programming as a tool for the general development of robot control systems which require the effective fusing of multiple information streams and a flexible and straightforward approach to task prioritisation and decomposition. The results further indicate that effective controllers can be developed to exploit information in systems, which are immersed in environments containing ambient communicated information. A possible application area would be passive sensing, where thermo-gradients, sound or disturbances in background noise can be used to discern information about the surroundings and those in it. The next chapter investigates the more deliberate (or active) use of communication, where the robots have to decide what information to communicate and when to do it.

# Chapter 7

# TASK III: KEEPING AN ENVIRONMENT SECURED

## 7.1 INTRODUCTION

The work presented here follows on from the findings in chapter 6 that the evolutionary process can benefit from the presence of communication. In this chapter, the focus is solely on the act of communication and reactions to it. A task is proposed in which the evolutionary process must decide when and how often to communicate as well as how to react to the receipt of communication. In this work, communication is the only tool available to the evolutionary process in order for it to improve the performance of the programs it produces. The chapter is structured as follows, section 7.2 introduces the task, section 7.3 lists and describes all the functions and terminals available for use in this work, section 7.4 presents the results and section 7.5 summarises the work.

## 7.2 TASK OVERVIEW AND IMPLMENTATION

The experimental parameters for this task are introduced in section 7.2.1, then as a means to define the context and relevancy of the task section 7.2.2 highlights the key properties and issues the task allows to be addressed and finally section 7.2.3 presents the details and implementation of the actual task.

### 7.2.1 Experimental parameters

Four task specific parameters are required by this task. These are:

- Number of robots 3
- Test suites size and number of test chosen. size 6 ; chosen 2
- Number of independent test runs. 4
- Duration before communicated information timed-out. 60 per robot

The settings of these parameters were determined as a result of a set of preparatory experiments. Each parameter was influenced by a different set of factors, these factors are discussed below.

The minimum number of robots required to perform the task was used, this was found to be three.

Time did not allow for all six tests to be run at once so in order to obtain similar effect but within an acceptable time frame two tests were randomly chosen from the six every generation. This was found, via experimentation to be effective.

Due to time constraints, only four independent runs could be produced.
A 60 t-state communication time-out duration per robot was used, this level was determined experimentally to be effective and offer encouragement towards persistent communication in a non-obtrusive manner.

### 7.2.2 Task overview

In brief, the task requires for the doors to an environment to be kept closed. It is the job of the robots to detect and close randomly opened doors. The task was constructed with two main objectives in mind, the integration of communication into existing systems and the relevancy of communicated information. The former can be generalised beyond this task and is achieved through the combined use of default/backup actions and state

based subsumption, while the latter is relatively specific to the task and is achieved via the existence of a dual approach to communication. Other issues and properties addressed by and exist in the task include:

- optimum use of communication
- competing information content
- multiple communication based accomplishment
- the need for high degree of co-operation to improve task completion
- robustness through noisy evaluation

The provision of a default controller (an existing or backup system) within the task allows the issue of integrating communication into existing systems to be tackled. This default controller defines the base set of actions, which are to occur if no communication-triggered action is stated. By allowing these actions to be overridden, communication can augment the system, the manner and degree to which this overriding takes place is totally under the control of the evolutionary process. This allows it to determine the circumstances under which this overriding should occur.

The task also addresses the relevancy of the information content communicated by providing two distinct uses for communication. Firstly, as a method to signify the use of code which extends the basic repertoire of the controller and secondly, to convey information. This allows the question of the need for task specific information content before the evolutionary process can make use of communication to be further investigated.

The issue of how best to use communication is addressed by providing a communication ADF, whose role is to determine when and what to communicate, to whom to communicate and under what circumstances.

In the task, there exist two forms of potentially useful information content that can be communicated. These forms compete against each other under evolutionary pressures to determine which one is most generally applicable to the task. This requires that the

evolutionary process determine the best way to use each of the information contents and evolve appropriate processing structures for each.

The task uses a state-based system to indicate the use of certain behavioural schemata. Multiple states are provided, each of which can be combined with others to effectively produce strategies which can aid in task completion. The evolutionary process must determine which combinations are best and what their respective information content requirement is for optimal performance.

The task was designed with the aim of increased co-operation leading to improved performance in mind. It is structured such that there is a minimum specification of two robots needed at the designated door before any progress can be made towards closing it. The rate at which the doors close is directly proportional to the number of robots involved, so the more robots that help the faster a door will close.

Noise is introduced into the controller evaluation process by changing the set of tests used to determine the fitness of a controller randomly every generation. This avoids overfitting the controllers to a specific test, encouraging robustness in the controllers.

Although the information content has a relevancy, this work shows that the act of communication in this task is more important. This duality within the communication and the ability of the evolutionary process to exploit it suggest that multiple roles for communication, beyond differing information content, offer greater room for flexibility within evolved controllers and the possibility of producing systems which apply communication in multiple and distinct roles. This is an issue addressed in the next chapter (chapter 8). Here the ability to determine when and what to communicate as well as how to respond to that communication are considered. The task is generalisable to one which requires co-ordinated and collective actions at time varying points in an environment. The task specific implementation details are given in the following section.

### 7.2.3 Task implementation

The task used to investigate the evolution of deliberate communication requires a collection of robots in a bounded environment to keep all access points into the environment closed. A 100 x 120 rectangular environment with a single access point (door) on each wall is used (the environment is shown in Figure 7.1). Doors are chosen to open at random, only one door can be open at any time, and only one robot is made aware of its opening by being given its id number. Every time a door opens its associated pressure setting is set to maximum. In order to close a door, at least two robots must be within docking distance of it, and then they must access its locking mechanism and keep hold of it until the door pressure falls to zero. The door will only be fully closed when this pressure reaches zero. The robot that was aware of the door being open now has its door open id negated. This locking of the door and other actions performed by the robots are specified in the form of schemata. The robots use states to determine which of the schema to activate. A set of the potentially useful states was made available for this task, these, along with a description, are given in Table 7.1. A default program utilising these states is used to control the basic motions of the robots. This program is designed primarily to keep the robot moving. It allows them to resolve collisions, wander, and close doors. However, the performance of this program in regard to the task is very poor, since a door will only ever be encountered by chance. All of the functions of this default program, with the exception of collision resolution, can be overridden. The evolutionary process is provided with two ADFs in order to achieve this. The first one, labelled procedure1 (#1) has access to all the communication software. The second one, labelled procedure2 (#2) has the ability to override the state of the default program. This procedure can only be triggered by communication. Regular communication is required if #2 is to be continuously executed. All communications time out after a fixed time period, at which point the data is blanked out with the lowest invalid value (see section 7.3.) A time-out value of 60 t-states per robot is used here. Figure 7.2 shows the default controller used by each of the robots in which both the default program and the role of the two ADFs can be seen. In Figure 7.2 the default program is shown in bold and it is split in two by calls to #1 and #2. The **Pri**

operators are used to allow the default program to be overridden (see section 7.3 for more information on the exact working of this operator as well as further information on the ADFs.)

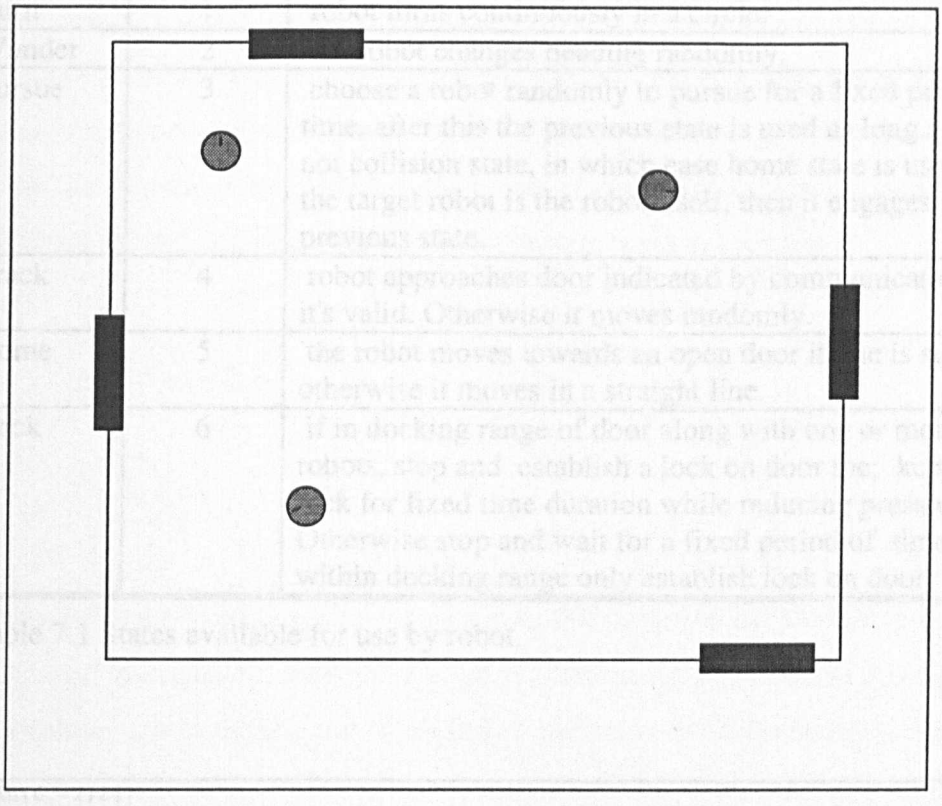


Figure 7.1 Test environment for task.

| STATE | STATE VALUE | DESCRIPTION |
|---|---|---|
| Collision | 0 | reverse and turn through random direction if collision detected otherwise move in a straight line. |
| Turn | 1 | robot turns continuously in a circle. |
| Wander | 2 | the robot changes heading randomly. |
| Pursue | 3 | choose a robot randomly to pursue for a fixed period of time, after this the previous state is used as long as it's not collision state, in which case home state is used. If the target robot is the robot itself, then it engages its previous state. |
| Track | 4 | robot approaches door indicated by communication if it's valid. Otherwise it moves randomly. |
| Home | 5 | the robot moves towards an open door if one is sighted otherwise it moves in a straight line. |
| Dock | 6 | if in docking range of door along with one or more other robots, stop and establish a lock on door the; keep this lock for fixed time duration while reducing pressure. Otherwise stop and wait for a fixed period of time, if within docking range only establish lock on door. |

Table 7.1 States available for use by robot.

```
main{0,-1}24[
     (IFGE (NoCl) (WST)          // initial collision check
        THEN (ClST)              // enter collision state
        ELSE
            (Pri
                 (Pri (#2) (#1))
                 (IFGE (NoCl) (WST)     // post ADF call collision test
                    THEN (ClST)
                    ELSE
                         (IFGE (Dok) (ClST)     // dockable range test
                            THEN (PwrST)        // enter dock state
                            ELSE (WST)          // enter wander state
                          )
                 )
            )
     )
]_Main
```

Figure 7.2 Default controller for robot with capacity to be overridden.

The task of the evolutionary process is to produce code for #1 and #2, which leads to improved performance in completing the task. Two constraints are used to evaluate the program:

1. the number of doors unclosed, measured in terms of percentage of overall pressure,
2. the time taken to close all the doors.

Constraint 1 is determined by tallying the amount of pressure reduced and subtracting it from the total possible amount, giving an indication as to the number of doors closed. A maximum number of 12 doors can open in a test. Constraint 2 is determined here by using the amount of energy left in the robots at the end of the test or once the last of the 12 doors is closed. These constraints can be expressed as follows:

$$f_r = (d*w) - \sum_{i=1}^{Rn} q_i, \qquad \text{Equation 7.1}$$

$$C_1 = \sum_{r=1}^{z} f_r / z. \qquad \text{Equation 7.2}$$

where Rn is the maximum number of robots taking part in each test, z is the number of tests used to evaluate a program, d is the maximum number of doors that can open in an environment, w is the amount of pressure required to close a single door, $q_i$ is the amount of pressure robot *i* contributed to closing doors in a single test, $f_r$ is effectively the amount of unclosed doors in test *r* and $C_1$ is constraint 1, the average amount of unclosed doors across the test cases.

Constraint 2 can be expressed as:

$$t_r = \sum_{i=1}^{Rn} e_i / (E*Rn), \qquad \text{Equation 7.3}$$

$$C_2 = \sum_{r=1}^{z} t_r / z. \qquad \text{Equation 7.4}$$

where $e_i$ is the amount of energy used by robot *i* at the end of a test, E is the maximum amount of energy available to each robot, $t_r$ is the percentage of total energy remaining at the end of each test and $C_2$ gives constraint 2, the average amount of time taken to close the doors.

By minimising these two constraints, the evolutionary system has the ability to produce effective controllers. The programs are tested in two environments, these being chosen randomly, at the end of each generation, from a suite of six possible scenarios. Each of these scenarios has different initial positions for the robots and different door opening sequences. The performance in each of the tests is averaged to give the overall fitness of a program. The tolerance level was set at [90 *] (the value 90 is for constraint 1 and * for constraint 2, where * means do not care). This allowed only programs with a constraint 1 value of less than 90 to be logged, in this respect any value for constraint 2 was acceptable. This level was adopted since it was found that on occasion a program using no communication could produce, as its best performance, a fitness vector of [91 99] (that's 91 for constraint 1 and 99 for constraint 2). By setting the tolerance level so, only communication based programs are logged.

## 7.3 FUNCTIONS AND TERMINALS

The GP used here makes use of three system blocks two ADFs and a main block. From section, 7.2 we know that the main block is constant and the ADFs undergo evolutionary change. In this section a list of the functions and terminals available to the evolutionary process are given along with a description of what they do (see Table 7.2.) A list of the functions and terminals used by each system block is also given. There exist 7 valid states which are given in Table 7.1. Two invalid state constants are employed throughout this work *LowInvalid* (-1) and *HighInvalid* (1000). These are returned by some functions to avoid changing the state of the robot, and are used as Boolean values by others. The values of the valid states are between these two values and can be seen in Table 7.1. The reasons behind the provision of these functions and terminals are as follows:

- To provide states which utilise communicated information internally and directly as well as others in which the use of information is controllable and unrestricted.

- To provide states which can be used individually or combined in a structured manner to accomplish the task using appropriate levels of communication and internal knowledge.
- To allow the robots to have an awareness of self and knowledge about the environment. This will allow them to fully explore the freedom offered by various state combinations.
- To allow the evolutionary process to decide how useful information content is to the task.

| FUNCTION & TERMINALS | DESCRIPTION |
|---|---|
| main | returns the state the robot is going to move into. Executes default program and calls #1 and #2. |
| procedure1 *-#1-* | decide when and how to communicate. Always Returns invalid state - *LowInvalid* -. |
| procedure2 *-#2-* | decides on and returns the state to override default program with. This ADF however can only be executed if some communication has been received. If no communication has been received then an invalid state is returned - *LowInvalid* - and the default program is not overridden. |
| IFGE **a b c d** | if **a** is greater than or equal to **b** then return **c** otherwise return **d**. |
| IFEq **a b c d** | if **a** is equal to **b** then return **c** otherwise return **d**. |
| Pri **a b** | if **a** is a valid state then return **a** otherwise return **b**. |
| See | return *HighInvalid* if an open door can be seen otherwise return *LowInvalid*. |
| NoCl | return *HighInvalid* if a collision detected otherwise return *LowInvalid*. |
| Dok | return *HighInvalid* if in docking range otherwise return *LowInvalid*. |
| PFound | return *HighInvalid* if PFound target encountered otherwise return *LowInvalid*. |
| DrOpen | return *HighInvalid* if aware of current open door otherwise return *LowInvalid*. |
| NoST | return an invalid state - *LowInvalid*. |
| LPut2 | transmit a fixed value (*HighInvalid*.) Returns invalid state - *LowInvalid*. |
| LPut3 | transmit internal information about open door. Returns invalid state -*LowInvalid*. |
| GetST | return communicated information received by robot. |
| CST | return robot's current state. |

| FUNCTION & TERMINALS | DESCRIPTION |
|---|---|
| WST | return wander state. |
| HmST | return home state. |
| ClST | return collision state. |
| PwrST | return dock state |
| PsuST | return pursue state. |
| TrkST | return track state. |
| TrnST | return turn state. |

Table 7.2 Available functions and terminals.

These functions and terminals are used by different system blocks, a list of these can be seen in Table 7.3.

| BLOCK | FUNCTIONS AND TERMINALS |
|---|---|
| main *FTset 24* | { IFGE, NoCl, Dok, Pri, #1, #2, WST, ClST, PwrST } |
| procedure1 *-#1- FTset 25* | { IFGE, IFEq, CST, WST, HmST, ClST, PwrST, LPut2, Lput3, DrOpen } |
| procedure2 *-#2- FTset 26* | { IFGE, IFEq, CST, WST, HmST, PwrST, PsuST, TrkST, TrnST, NoCl, Dok, PFound, DrOpen, See, GetST, NoST } |

Table 7.3 Functions and terminals accessible to individual system blocks.

## 7.4 EXPERIMENTS AND ANALYSIS

*Aim:* To establish whether the evolutionary process can determine: when, what and how often to communicate to facilitate task completion.

In this experiment three robots were used, which allowed the evolutionary process a wide variety of potential strategies without trivialising the endeavour. Global communication between the robots was used and two forms of communicated information were available for their use. The results were generated by running the GP four times using different initial random seed values for both the GP and the interpreter.

The distributed evaluation method was employed using four HP-700 series workstations to help distribute the load.

### 7.4.1 Results

The controllers presented here are those whose general form was common to all 4 runs and which exhibited some noteworthy traits. Each run took between 2 and 3 days to complete and produced large amounts of tolerable solutions. The average time taken to find the first tolerable solution was 31 generations. The fitness of these initial solutions was poor and the strategy they employed unrobust, leading to tolerable performance in only single specific test case combinations. The typical average fitness of these controllers was [100 99] and the average best fitness [89.5 99]. The average time taken to produce the best solution was 78 generations. These best controllers had a worst average fitness of [88 99] and a best average fitness of [75 99].

A varied selection of controllers was produced by the evolutionary process. Due to the dynamic nature of the testing (random selection each generation), controllers emerged which were proficient only in certain test environments. Controllers with good all round performance were also produced. To show this and the effectiveness of the controllers, for each controller included, a range of fitnesses is given which represents its performance envelope for all the possible combinations of tests. This range is expressed as follows [[a b] $\leftrightarrow$ [c d]], which means, that the fitness for the controller ranged from a b to c d. where **a** & **c** are values for constraint 1 and **b** & **d** are values for constraint 2. As a way of providing a benchmark, the performance range of the default controller alone was determined, which was [[100 99] $\leftrightarrow$ [93 99]]. This tells us that constraint 2 remained constant at 99 and constraint 1 varied from 100 to 93. The controllers evolved fell into one of two possible communication and execution groups. These groupings are given below:

Communication

i. constant updating of communicated information (continuous transmission.)

ii. conditional updating of communicated information (intermittent transmission.)

Execution

i. executes default program in all situations.

ii override one or more aspects of the default program with the exception of collision resolution.

These groups in most cases can be further sub-divided along lines of knowledge and communication dependency. The benefit of communicating was discovered very early on and all the tolerable solutions produced used it. Its main use was to trigger the execution of *procedure2.* Controllers were produced which also utilised the information communicated. Four examples of such controllers can be seen in Figures 7.3 to 7.6. In the first of these, Figure 7.3, the use of the type of communicated information produces a controller, which is effective at the task. If the LPut2 (a robot transmits a fixed value) communication operator was used instead of LPut3 (a robot transmits its internal information regarding open door) the controller would be totally ineffective since it will remain solely in the home state. As well as this obvious use there is also a more subtle role played by communicated information which increases the chances of the robot going into the dock state, for, if a robot receives communication from a robot which knew about a previously open door, then the robot will remain in the power state until it gets communication from a robot which knows about the present open door. This should lead to worse performance

```
system[
    procedure1{0,1}25[ (IFEq (LPut3) (CST) THEN (CST) ELSE (LPut3))]_Proc

    procedure2{0,1}26[ (IFGE (GetST) (Dok) THEN (HmST) ELSE (PwrST))]_Proc

    main{0,-1}24[ (IFGE (NoCl) (WST)
                    THEN (ClST)
                    ELSE
                        (Pri
                            (Pri (#2) (#1))
                            (IFGE (NoCl) (WST)
                               THEN (ClST)
                               ELSE (IFGE (Dok) (ClST) THEN (PwrST) ELSE (WST))
                            )
                        )
                )
      ]_Main
  ]_Sys  [[83 99]  ↔ [66 99]]
```

Figure 7.3 Communication dependent controller.

but it produces one of the lowest scoring (i.e. best) controllers. This performance level could result from the fact that fewer robots participate in the door closing process so the chances of them interfering with each other is reduced. In the second example (Figure 7.4)

the use of the communicated information transmitted by LPut2 prevents the default controller from being overridden. If, LPut3 was used instead, the overriding of the controller would be intermittent and dependent on the knowledge of communicating robots. When the default controller is overridden the door id code received will be used to determine the state of the robot. This would result in an ineffective controller. In the third example (Figure 7.5) communicated information is used to identify which door to approach. This is achieved with the track state and LPut3 communication. On receipt of open door id then the door is approached, otherwise the robot moves randomly. If LPut2 was used here, the robots will continuously move randomly producing an ineffective controller. The final example (Figure 7.6) uses conditional transmission of LPut3 information to improve the performance of a track state based controller. This results in an effective all round controller. In all these examples the evolutionary process was capable of matching individual controllers, communicated information type (*procedure1*) with its information usage (*procedure2*). The process was also capable of producing controllers

```
system[
    procedure1{0,1}25[
         (IFGE (LPut2) (PwrST)
            THEN (IFGE (DrOpen) (LPut2) THEN (ClST) ELSE (PwrST))
            ELSE (WST)
          )
        ]_Proc

    procedure2{0,1}26[ (IFGE (NoST) (Dok) THEN (HmST) ELSE (GetST))]_Proc

    main{0,-1}24[ (IFGE (NoCl) (WST)
                      THEN (ClST)
                      ELSE
                           (Pri
                               (Pri (#2) (#1))
                               (IFGE (NoCl) (WST)
                                  THEN (ClST)
                                  ELSE (IFGE (Dok) (ClST) THEN (PwrST) ELSE (WST))
                               )
                           )
                  )
      ]_Main
  ]_Sys  [[95 99]  ↔ [83 99]]
```

Figure 7.4 Communication dependent controller employing communicated information to allow default program to be executed.

which changed the communication type used, an example of this is shown in Figure 7.7. Here it uses the internal knowledge of the robot to determine the type of communication to use. Only those robots aware of an open door use LPut3 and all the others use LPut2. This controller exhibits effective all round performance. All the aforementioned controllers use continuous communication and those which utilise LPut3 owe their performance level to

this. Further examples of these controllers can be seen in Figures 7.8 to 7.15. Some intermittent transmitters were also produced, examples of which can be seen in Figures 7.6 and 7.16. In Figure 7.6, the internal knowledge of the robot as well as its current state are used to determine when it should transmit information. The controller in Figure 7.16 applies a similar technique, communication only taking place when the current open door id is less than the value associated with the home state.

```
system[
    procedure1{0,1}25[
         (IFGE (LPut3) (ClST)
            THEN (IFGE (WST) (DrOpen) THEN (PwrST) ELSE (WST))
            ELSE (WST)
          )
       ]_Proc

    procedure2{0,1}26[ (IFGE (Dok) (CST) THEN (DrOpen) ELSE (TrkST))]_Proc

    main{0,-1}24[ (IFGE (NoCl) (WST)
                      THEN (ClST)
                      ELSE
                           (Pri
                                (Pri (#2) (#1))
                                (IFGE (NoCl) (WST)
                                   THEN (ClST)
                                   ELSE (IFGE (Dok) (ClST) THEN (PwrST) ELSE (WST))
                                )
                           )
                  )
       ]_Main
  ]_Sys  [[95 99]  ↔ [83 99]]
```

Figure 7.5 Controller using communication dependent state for co-ordination.

```
system[
    procedure1{0,1}25[ (IFGE (DrOpen) (CST) THEN (LPut3) ELSE (DrOpen))]_Proc

    procedure2{0,1}26[ (IFGE (TrnST) (Dok) THEN (TrkST) ELSE (DrOpen))]_Proc

    main{0,-1}24[ (IFGE (NoCl) (WST)
                      THEN (ClST)
                      ELSE
                           (Pri
                                (Pri (#2) (#1))
                                (IFGE (NoCl) (WST)
                                   THEN (ClST)
                                   ELSE (IFGE (Dok) (ClST) THEN (PwrST) ELSE (WST))
                                )
                           )
                  )
       ]_Main
  ]_Sys  [[87 99]  ↔ [75 99]]
```

Figure 7.6 Controller exhibiting knowledge dependent communication and state co-ordination.

```
system[
    procedure1{0,1}25[ (IFGE (DrOpen) (ClST) THEN (LPut3) ELSE (LPut2))]_Proc

    procedure2{0,1}26[
         (IFGE (CST) (Dok)
            THEN (IFGE (GetST) (NoST) THEN (HmST) ELSE (Dok))
            ELSE (NoCl)
          )
       ]_Proc

    main{0,-1}24[ (IFGE (NoCl) (WST)
                     THEN (ClST)
                     ELSE
                          (Pri
                              (Pri (#2) (#1))
                              (IFGE (NoCl) (WST)
                                 THEN (ClST)
                                 ELSE (IFGE (Dok) (ClST) THEN (PwrST) ELSE (WST))
                               )
                           )
                  )
       ]_Main
  ]_Sys  [[87 99]  ↔ [80 99]]
```

Figure 7.7 Controller employing knowledge to determine communication type.

The initial effective controllers produced relied on the default controller for assistance. Examples of these can be seen in Figures 7.4 to 7.9, 7.14, 7.16. Later controllers tended to override it completely, which lead in some cases to better performance. The controller in Figure 7.8 is particularly interesting. It uses all the facilities offered by the default controller, only subsuming it when a robot can see an open door and the robot is not within the docking range of it. Under these circumstances, the home state is used. This gives rise to a controller which wanders around until it can see an open door, then it switches to home state to approach it. A similar strategy can also be seen in Figure 7.7. All the other examples make use of the default program for docking only. The evolutionary process found advantage in overriding all possible operations offered by the default program. Testing for the docking distance within *procedure2* allowed a robot to dock earlier and made it less sensitive to collisions increasing the likelihood of successfully docking despite being involved in a collision within the docking area. By using either the home, track or turn states to find the doors a more effective and reliable find strategy was produced. Examples of these controllers can be seen in Figures 7.3, 7.10 to 7.13 and 7.15. In Figure 7.11 the controller uses two states, home state (to find the door) and turn state (to avoid any late collisions). The example shown in Figure 7.13 does not allow the robot with knowledge of the open door to enter the docking state. The controller in Figure 7.10 relies partly on the default program. If the docking range is entered after the first test then the default program is used for docking.

```
system[
    procedure1{0,1}25[ (IFGE (PwrST) (CST) THEN (LPut3) ELSE (CST))]_Proc

    procedure2{0,1}26[
         (IFGE (HmST) (Dok)
            THEN (IFGE (See) (PsuST) THEN (HmST) ELSE (NoST))
            ELSE (DrOpen)
          )
       ]_Proc

    main{0,-1}24[ (IFGE (NoCl) (WST)
                      THEN (ClST)
                      ELSE
                           (Pri
                               (Pri (#2) (#1))
                               (IFGE (NoCl) (WST)
                                  THEN (ClST)
                                  ELSE (IFGE (Dok) (ClST) THEN (PwrST) ELSE (WST))
                               )
                           )
                  )
       ]_Main
  ]_Sys  [[91 99]  ↔ [79 99]]
```

Figure 7.8 Controller showing state dependent communication with visual dependent co-ordination.

```
system[
    procedure1{0,1}25[
         (IFGE (LPut3) (HmST)
            THEN (LPut3)
            ELSE (IFGE (PwrST) (CST) THEN (LPut3) ELSE (LPut3))
          )
       ]_Proc

    procedure2{0,1}26[ (IFGE (NoST) (Dok) THEN (HmST) ELSE (DrOpen))]_Proc

    main{0,-1}24[ (IFGE (NoCl) (WST)
                      THEN (ClST)
                      ELSE
                           (Pri
                               (Pri (#2) (#1))
                               (IFGE (NoCl) (WST)
                                  THEN (ClST)
                                  ELSE (IFGE (Dok) (ClST) THEN (PwrST) ELSE (WST))
                               )
                           )
                  )
       ]_Main
  ]_Sys  [[91 99]  ↔ [79 99]]
```

Figure 7.9 Controller exhibiting knowledge dependent docking.

```
system[
    procedure1{0,1}25[ (IFGE (LPut2) (HmST) THEN (WST) ELSE (ClST))]_Proc

    procedure2{0,1}26[
         (IFGE (PwrST) (Dok)
            THEN (IFGE (NoST) (Dok) THEN (HmST) ELSE (NoST))
            ELSE (PwrST)
          )
       ]_Proc

    main{0,-1}24[ (IFGE (NoCl) (WST)
                     THEN (ClST)
                     ELSE
                          (Pri
                              (Pri (#2) (#1))
                              (IFGE (NoCl) (WST)
                                 THEN (ClST)
                                 ELSE (IFGE (Dok) (ClST) THEN (PwrST) ELSE (WST))
                               )
                           )
                  )
       ]_Main
  ]_Sys  [[91 99]  ↔ [75 99]]
```

Figure 7.10 Controller employing last minute override avoidance check.

```
system[
    procedure1{0,1}25[ (IFGE (LPut2) (HmST) THEN (CST) ELSE (CST))]_Proc

    procedure2{0,1}26[
         (IFGE (HmST) (Dok)
            THEN (IFGE (NoST) (NoCl) THEN (HmST) ELSE (TrnST))
            ELSE (PwrST)
          )
       ]_Proc

    main{0,-1}24[ (IFGE (NoCl) (WST)
                     THEN (ClST)
                     ELSE
                          (Pri
                              (Pri (#2) (#1))
                              (IFGE (NoCl) (WST)
                                 THEN (ClST)
                                 ELSE (IFGE (Dok) (ClST) THEN (PwrST) ELSE (WST))
                               )
                           )
                  )
       ]_Main
  ]_Sys  [[91 99]  ↔ [79 99]]
```

Figure 7.11 Controller exhibiting additional collision resolution behaviour.

```
system[
    procedure1{0,1}25[ (IFGE (LPut2) (HmST) THEN (DrOpen) ELSE (LPut2))]_Proc

    procedure2{0,1}26[ (IFGE (HmST) (Dok) THEN (HmST) ELSE (PwrST))]_Proc

    main{0,-1}24[ (IFGE (NoCl) (WST)
                     THEN (ClST)
                     ELSE
                          (Pri
                              (Pri (#2) (#1))
                              (IFGE (NoCl) (WST)
                                 THEN (ClST)
                                 ELSE (IFGE (Dok) (ClST) THEN (PwrST) ELSE (WST))
                               )
                           )
                  )
       ]_Main
  ]_Sys  [[83 99]  ↔ [66 99]]
```

Figure 7.12 Highly effective controller overriding all possible default program actions.

```
system[
    procedure1{0,1}25[
          (IFGE (LPut2) (LPut3)
             THEN (LPut3)
             ELSE (IFEq (PwrST) (WST) THEN (CST) ELSE (LPut3))
          )
       ]_Proc

    procedure2{0,1}26[ (IFGE (DrOpen) (Dok) THEN (HmST) ELSE (PwrST))]_Proc

    main{0,-1}24[ (IFGE (NoCl) (WST)
                      THEN (ClST)
                      ELSE
                           (Pri
                               (Pri (#2) (#1))
                               (IFGE (NoCl) (WST)
                                  THEN (ClST)
                                  ELSE (IFGE (Dok) (ClST) THEN (PwrST) ELSE (WST))
                               )
                           )
                  )
       ]_Main
  ]_Sys  [[100 99]  ↔ [75 99]]
```

Figure 7.13 Controller exhibiting knowledge dependent participation.

```
system[
    procedure1{0,1}25[ (IFGE (CST) (CST) THEN (LPut3) ELSE (LPut2))]_Proc

    procedure2{0,1}26[
          (IFGE (WST) (Dok)
             THEN
                  (IFGE
                      (IFGE
                          (IFEq (TrnST) (Dok)
                             THEN (NoST)
                             ELSE (PFound)
                          )
                          (Dok)
                        THEN (See)
                        ELSE (IFGE (PFound) (PFound) THEN (PsuST) ELSE (HmST))
                      )
                     (PsuST)
                   THEN (HmST)
                   ELSE (See)
                  )
             ELSE (DrOpen)
          )
       ]_Proc

    main{0,-1}24[ (IFGE (NoCl) (WST)
                      THEN (ClST)
                      ELSE
                           (Pri
                               (Pri (#2) (#1))
                               (IFGE (NoCl) (WST)
                                  THEN (ClST)
                                  ELSE (IFGE (Dok) (ClST) THEN (PwrST) ELSE (WST))
                               )
                           )
                  )
       ]_Main
  ]_Sys  [[91 99]  ↔ [79 99]]
```

Figure 7.14 Large controller giving relatively good performance.

```
system[
     procedure1{0,1}25[
          (IFEq (LPut2) (CST)
             THEN (IFEq (ClST) (PwrST) THEN (PwrST) ELSE (ClST))
             ELSE (WST)
          )
       ]_Proc

     procedure2{0,1}26[
          (IFGE (HmST) (Dok)
             THEN (IFGE (DrOpen) (NoST) THEN (HmST) ELSE (PwrST))
             ELSE (PwrST)
          )
       ]_Proc

     main{0,-1}24[ (IFGE (NoCl) (WST)
                      THEN (ClST)
                      ELSE
                           (Pri
                                (Pri (#2) (#1))
                                (IFGE (NoCl) (WST)
                                   THEN (ClST)
                                   ELSE (IFGE (Dok) (ClST) THEN (PwrST) ELSE (WST))
                                )
                           )
                   )
       ]_Main
  ]_Sys  [[91 99]  ↔ [75 99]]
```

Figure 7.15 Controller exhibiting knowledge dependent participation.

```
system[
    procedure1{0,1}25[ (IFGE (HmST) (DrOpen) THEN (LPut2) ELSE (WST))]_Proc

    procedure2{0,1}26[ (IFGE (NoST) (Dok) THEN (HmST) ELSE (DrOpen))]_Proc

    main{0,-1}24[ (IFGE (NoCl) (WST)
                      THEN (ClST)
                      ELSE
                           (Pri
                                (Pri (#2) (#1))
                                (IFGE (NoCl) (WST)
                                   THEN (ClST)
                                   ELSE (IFGE (Dok) (ClST) THEN (PwrST) ELSE (WST))
                                )
                           )
                   )
       ]_Main
  ]_Sys  [[83 99]  ↔ [79 99]]
```

Figure 7.16 Controller displaying knowledge dependent communication and docking.

### 7.4.2 Discussion of results

The production and utilisation of controllers employing communication is evident, performance improvements are achieved with better override and communication structures. The evolution of such communication-based controllers was not explicitly directed, instead it was a potential avenue available to the evolutionary process in order to improve performance. Therefore, because of this unrestricted route, a varied and distinct set of solutions was produced. For the number of robots and the schema coding, the optimal fitness value is [66 99]. With the constraints of unoptimised schemas, the evolutionary process still produced optimal controllers (see Figures 7.3 and 7.12 as well as

Table 7.4 for an overall summary of the performance of the controllers considered in this chapter.) Several other controllers were also produced which exhibited tolerable performance over all combinations of test cases (Figures 7.6,7.7,7.12 and 7.16.) The optimal usage of communication is achieved regardless of the type of information, which suggests one of two things. Firstly that the act of communication is beneficial. Secondly, information content can be exploited when essential and inessential to task. Although the structure of the code in some cases may be similar, the semantics are specific to the type of communication used. In some cases, the usage of communicated information does not add anything to the performance, an example of this is the controller in Figure 7.7. Several controllers were produced which replaced the *GetSt* operator with a *NoSt* operator, resulting in a controller with identical fitness. Many of the effective controllers used communication ADFs that had some structure to them. Although in many cases these ADFs would continuously communicate, they never reduced to a single operator call. This would be considered the optimal continuous transmission form. There seemed to be a minimal time restriction between executions of a robots override ADF in order for optimal performance to be realised. The success of the intermittent communication can then in part be attributed to this as well as other factors. These include the ability of discerning the level of acceptable timing out of the communicated information and the ability to establish the number, and in some cases minimum number, of robots to apply to the task.

The time out duration places an implicit program length restriction for development of optimal controllers. The longer this value, the larger the communication freedom a program has. If this value is reduced then the potential usefulness of intermittent communication will diminish and the execution lengths of the continuous transmission ADFs would have to be reduced.

The range and variety of the controllers evolved was not actually stated or encouraged through the fitness function, even though such solutions were produced. This can be attributed to the use of dynamic testing (changing test cases every generation). Apart from myriad solutions, its use also leads to more robust controllers, which offer a greater level of generalisation. Further, its use also aids the development of diverse solutions and facilitates the growth of niches, which are controllers, which perform well

under certain circumstances of environmental conditions. These niches help preserve and encourage a diversity of solutions as well as prevent or delay premature convergence.

| CHAPTER 7 FIGURE REFERENCE | SUMMARY OF CONTROLLER | BEST FITNESS | | WORST FITNESS | |
|---|---|---|---|---|---|
| | | constraint 1 | constraint 2 | constraint 1 | constraint 2 |
| 2 | Default controller alone. | 93 | 99 | 100 | 99 |
| 3 | One of best performers, utilises communicated information and internal knowledge to determine degree of participation, uses complex communication (override). | 66 | 99 | 83 | 99 |
| 4 | Some of default controllers use avoided due to use of communication (partial). | 83 | 99 | 95 | 99 |
| 5 | Communication dependent co-ordination state used (partial). | 83 | 99 | 95 | 99 |
| 6 | Knowledge dependent communication (partial). | 75 | 99 | 87 | 99 |
| 7 | Knowledge determines communication type (partial). | 80 | 99 | 87 | 99 |
| 8 | State dependent communication and visual dependent co-ordination used (partial). | 79 | 99 | 91 | 99 |

| CHAPTER 7 FIGURE REFERENCE | SUMMARY OF CONTROLLER | BEST FITNESS | | WORST FITNESS | |
|---|---|---|---|---|---|
| | | constraint 1 | constraint 2 | constraint 1 | constraint 2 |
| 9 | Multiple communication calls ensuring constant triggering of *procedure2* (partial). | 79 | 99 | 91 | 99 |
| 10 | Last minute override avoidance check (override). | 75 | 99 | 91 | 99 |
| 11 | Additional collision resolution strategy employed (override). | 79 | 99 | 91 | 99 |
| 12 | One of the best controllers, uses simple communication (override). | 66 | 99 | 83 | 99 |
| 13 | Knowledge dependent participation (override). | 73 | 99 | 100 | 99 |
| 14 | Large controller (partial). | 79 | 99 | 91 | 99 |
| 15 | Knowledge dependent participation (override). | 75 | 99 | 91 | 99 |
| 16 | Knowledge dependent communication (partial). | 79 | 99 | 87 | 99 |

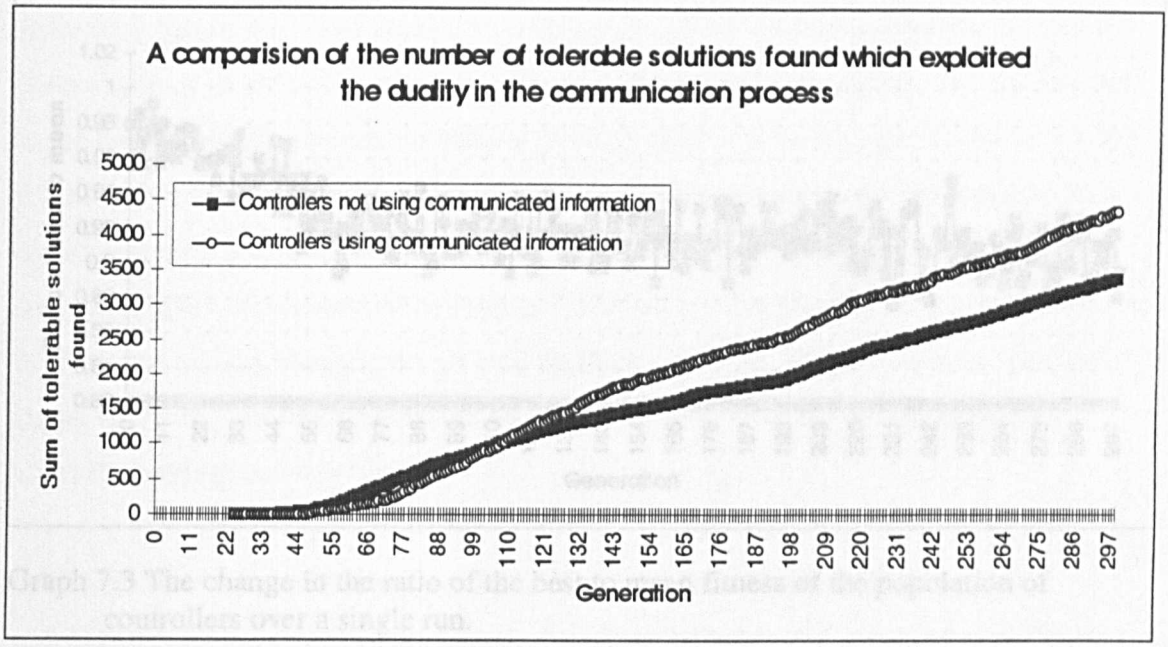
Table 7.4 Summary table of performance of the evolved controllers reported in this chapter.

Graph 7.1 shows two plots, one representing the cumulated total of tolerable ICOP controllers (use communicated information received) evolved and the other the cumulated total of non-ICOP controllers (do not use communicated information received). The non-ICOP based solutions were found first and their numbers increased

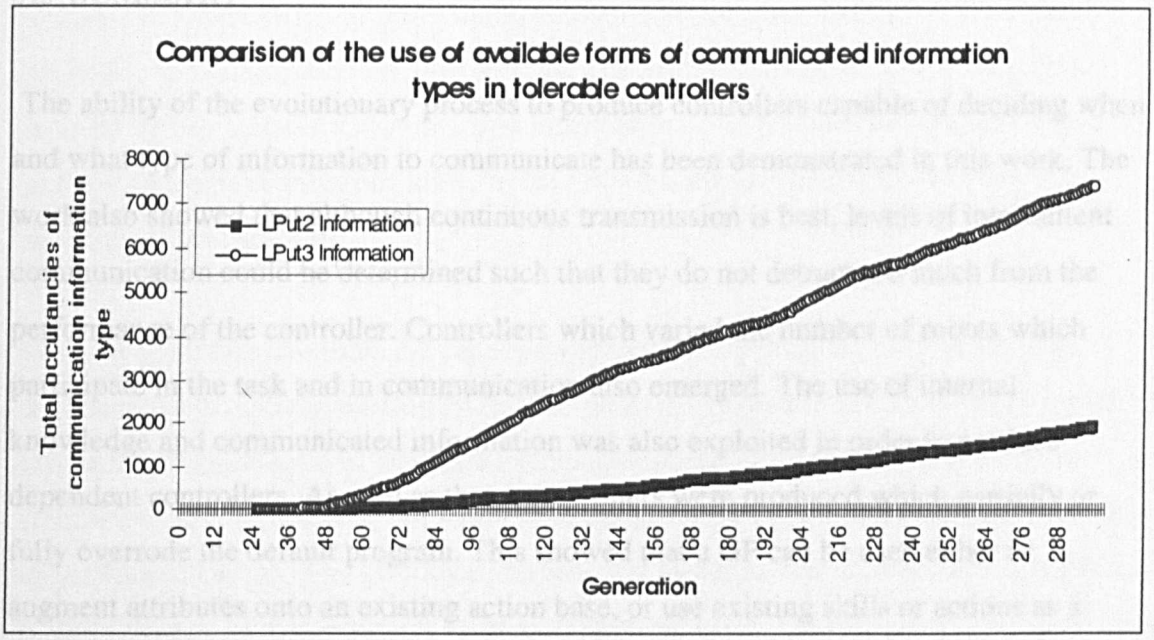
faster at the beginning, however after about 120 generations, the number of ICOP based controllers found over took them and continue to increase faster. This indicates that a quick, simple and effective use for just communicating was found early, however it was a long time before any appropriate strategies that used the communicated information were devised. Once the strategies were found, the benefits of using communicated information offered increased possibilities for controllers. Although the benefit of using communicated information caused a faster increase in the number of ICOP based controllers, it did not reduce the number of non-ICOP solutions, just that their rate of increase was less. This indicates that the act of communication offers benefits, and the relatively small disparity between the two curves indicates these benefits are long lasting.

Graph 7.2 shows two plots, each representing the respective cumulative total of tolerable solutions produced using each of the two available communication operators. From this graph, it can be seen that the preferred information content was that of LPut3, which allows task dependant information about which door is open to be conveyed. The use of this information vastly outstrips that of LPut2, which is the transmission of simple non-task dependant information. This suggest that communication of task dependant information is more beneficial to the task and, from Graph 7.1 that the use of such information allows for the emergence of greater number of controllers which actually find a use for the communicated information (this latter point comes from the increase in the use of communicated information shown in Graph 7.1). If the information being communicated were not being used, it would be expected that the two curves in Graph 7.2 would be more or less the same (in fact maybe even the simplicity of LPut2 would make it slightly more popular), the fact there is a significant difference in favour of LPut3 contradicts this.

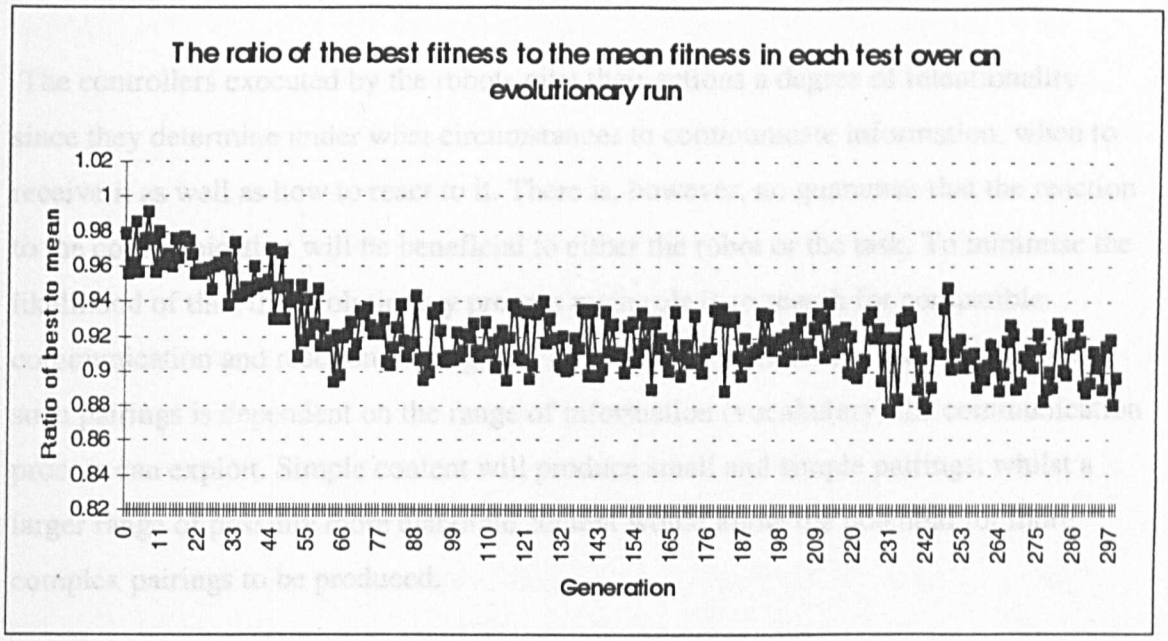
Graph 7.3 shows the ratio of best fitness to the mean fitness over the length of a run. The plot start at about 0.98 and over 60 generation period drops to 0.92 and then varies within a small a narrow band (0.94 to 0.9) for the rest of the run. This suggests a steady improvement in the best and the average of population, also that regardless of the test case the ratio of the generation best to generation mean is always comparable.




Graph 7.1 A comparison between the number of controllers evolved which did and didn't use communicated information.




Graph 7.2 A comparison of the number of controllers evolved which used either task dependent or independent.




Graph 7.3 The change in the ratio of the best to mean fitness of the population of controllers over a single run.

## 7.5 SUMMARY

The ability of the evolutionary process to produce controllers capable of deciding when and what type of information to communicate has been demonstrated in this work. The work also showed that although continuous transmission is best, levels of intermittent communication could be determined such that they do not detract too much from the performance of the controller. Controllers which varied the number of robots which participate in the task and in communication also emerged. The use of internal knowledge and communicated information was also exploited in order to produce dependent controllers. As well as these, controllers were produced which partially or fully overrode the default program. This showed that a GP can be used either to augment attributes onto an existing action base, or use existing skills or actions as a back system which is subsumed by more intricate or alternate modes of actions. In this respect, the use of communication by the evolved controllers in this chapter can be viewed as either a method to allow default or back up actions to exist and only come into play under certain specified situations, or as way of guaranteeing a minimal level of performance should communication fail.

The controllers executed by the robots give their actions a degree of intentionality, since they determine under what circumstances to communicate information, when to receive it as well as how to react to it. There is, however, no guarantee that the reaction to the communication will be beneficial to either the robot or the task. To minimise the likelihood of this, the evolutionary process main role is to search for compatible communication and reaction pairings (#1 and #2). The availability and formation of such pairings is dependent on the range of information (vocabulary) the communication process can exploit. Simple content will produce small and simple pairings, whilst a larger range of possibly more elaborate content would allow the potential for more complex pairings to be produced.

The work showed that there was a distinction between communicating and communicating information. The subtle difference between the two can be highlighted if both of these capabilities are restated in terms based on the latter. The former becomes, the benefit derived from the receiving communication is sufficient in itself to obviate the need to use the information contained in the communication. The latter becomes, an additional benefit beyond that offered by communicating alone can be realised by utilising the communicated information. The evolutionary process was able to draw adequate distinctions between the two of these and evolve controllers appropriate for each case.

This work further identifies the capability of the genetic programming tool to add functionality to already existing systems or build up a system from a set of pre-specified emergency/default actions. This latter point is important in the development of safety critical systems, in that they are required to demonstrate a level of predictable behaviour in the face of catastrophic failures in higher level systems. The former point allows the useful life span of robot systems to be extended and offers them the ability to adapt their functionality to their environment, if they find it ineffective.

In closing this chapter, a final brief reiteration of how the results translate to the wider field of robotic research is presented. The work shows a method for allowing a level of

predictable behaviour in systems in the face of a major failure of high-level systems, it also offers a way of extending and adapting the functionality of systems allowing for post diagnostic improvements to be made. Finally, it shows that in systems where active communication is necessary (e.g., sonar based detection and night vision systems) the most effective communication method for transmitting the most appropriate information can be identified and implemented. As well as this it could determine if there was a need for relevant information in the communication (i.e. is communication to be used to trigger other sub-systems within the existing system. For example in sonar based detection, the returning echo of previously emitted sonar waves is used to trigger other systems, or is it to be used to convey information directly to it). The next chapter extends these findings and investigates the ability of the GP to distinguish between and exploit multiple classes of communication and directable communications.

# Chapter 8

# TASK IV: CIRCUIT COMPLETION

## 8.1 INTRODUCTION

The communication-based results presented so far have shown that the evolutionary process can take advantage of the presence of communication and formulate communication and action systems appropriate to the task and to the type of communication available. In this, the penultimate chapter, the GP's ability as a method for direct control is investigated. In all the previous experiments, information was communicated between the robots. This information could be ignored or processed in a fashion determined by the robot. In the work of this chapter, two classes of communication are present, state information and control instructions. The state information will only be transmitted when requested (this is equivalent to the communication used in chapters 6 and 7). The control instructions cannot be ignored and are targeted at individuals. These cause a direct change of behaviour. Genetic programming is used to evolve controllers capable of completing a simple circuiting task for 6 robots. This task is covered in detail in section 8.2 and all relevant functions and terminals are presented in section 8.3. Section 8.4 reports the experiments undertaken and results obtained and section 8.5 draws conclusions on the work.

## 8.2 TASK OVERVIEW AND IMPLEMENTATION

Section 8.2.1 presents the experimental parameters used in this task, section 8.2.2 discusses the properties of the task and how they relate to the aim of evolving controllers which utilise communication to exchange information and transmit control signals. The task and its associated implementation details are presented in full in section 8.2.3.

### 8.2.1 Experimental parameters

This task makes use of four task specific experimental parameters. These and their associated value are:

- Number of robots 6
- Test suites size and number of test chosen. size 16 ; chosen 2
- Number of independent test runs. 3
- Duration before communicated information timed-out. 45

The values of these parameters were determined as a result of a set of preparatory experiments. The factors that influenced the levels of these parameters are presented below.

Six robots are used here, the large number was found to be required in order to encourage the development of effective controllers (i.e. more interactive between robots) and offer a wide range of possible communication structures.

Time did not allow for more than two tests to be run at a time, so in order to best approximate the desired effect, two tests were chosen randomly from a test suite of sixteen. The reason for the large test suite size was due to the large number of robots, resulting in a greater number of distinctive relative positions between robots.

The large number of robots used further curtailed the number of independent runs possible. As a result of time restrictions only 3 runs could be undertaken.

A 45 t-state communication time-out duration per robot was used. This level was determined so as to avoid continuous responding to last communicated instruction (obviating the need for a "stop doing last instruction" communications to be sent once it believed the other robot has had enough time to perform instructions as well as preventing an increase in unnecessary communication traffic) and also as a way that

encourages the use of communication by introducing a time factor of relevancy. The value of this duration was determined experimentally and was a factor of the average length of a typical program.

### 8.2.2 Task overview

In brief, the task requires for a number of robots on a finite series on concentric tracks to maximise the distance they travel and minimise the amount of time spent involved in collisions within a defined time period. The task was constructed to allow for the investigation of the performance of the evolutionary system in evolving controllers, when multiple roles for communication are available. The generaliseability of the task to that of maximising the use of a finite shared resource by a finite set a competing individuals, indicates that it encapsulates the need for co-ordinated and informed actions by individuals. As such two distinct areas of potential application for communication exist, a method for co-ordination (control instructions) and as a means of being informed (information gathering). As well as the multiple roles for communication, the task also encompasses the following properties:

- directable communication
- multiple communication based accomplishment
- large robot population with large and noisy evaluation process.

In order to offer maximum flexibility in the gathering of information and co-ordination of activities, the two forms of communication were provided with an approximate group style communication and a one-to-one style communication. This directablility of communication allows for the development of controllers that use communication in a highly efficient manner.

The existence of multiple roles for communication and their associated operator base results in a large number of communication based strategies which will have varying degrees of success in achieving the task. The ability to determine in which role to apply

which form of communication as well as where and when offers a substantial challenge in strategy development for the evolutionary process.

A relatively large number of robots are used in this task. This is to increase the likelihood of the robots encountering each other and hence the need for an effective and general strategy. The development of a general as well as robust strategy is further encouraged via the use of a noisy evaluation process. For this, every generation a random and small set of test cases is chosen from a large set of possible test cases. The full specification and implementation details of the task are given in the following section.

### 8.2.3 Task implementation

Genetic programming is to be used to produce controllers which decide when and what information to request and from whom, and also when and how to control the actions of other robots. A collection of functions and terminals provided for these are given in section 8.3. The task adopted here is a simple one. It requires a number of robots to move as far as possible around a set of pre-defined tracks or circuits and avoid collisions. There are 16 possible test cases available for use in which the robots have different headings, positions and initial track numbers. Two test cases are randomly chosen from the available list every generation which are used to evaluate the current population (there are two distinct shapes of tracks available to choose and these can be seen in Figures 8.1 and 8.2). In each test case, two tracks are available for use, an inner and an outer one. These tracks are continuous and polygonal in shape. Six robots are used, each of which is running the same program. Each of the robots has a unique identification number, which must be used to direct communication to them. Functions are provided to enable specific one-to-one communication as well as general group communication (here members of the group are communicated with one at a time in a sequential manner). An individual also has the capability to communicate with itself but this can only occur using the one-to-one communication.

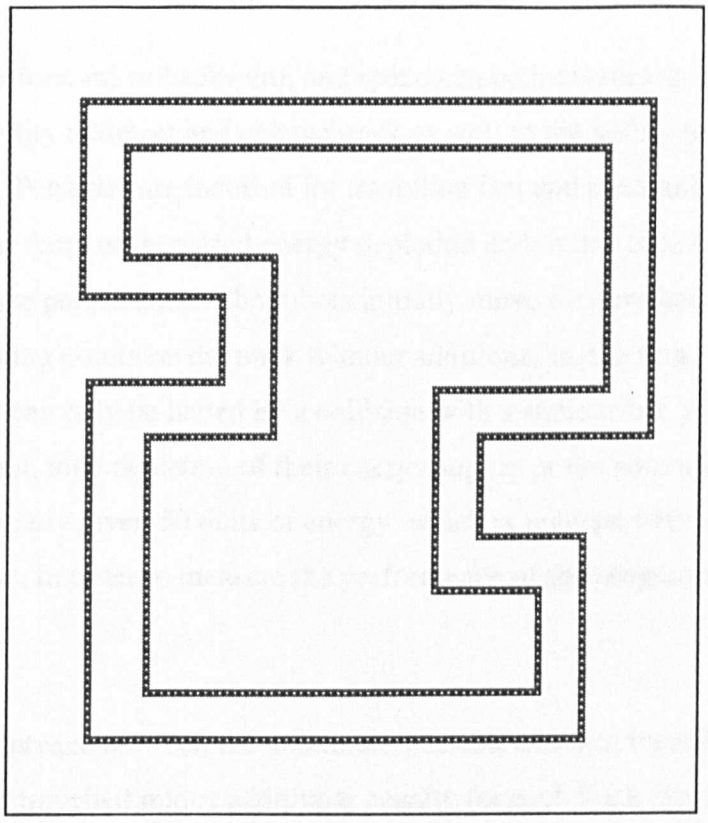


Figure 8.1 Test environment 1.

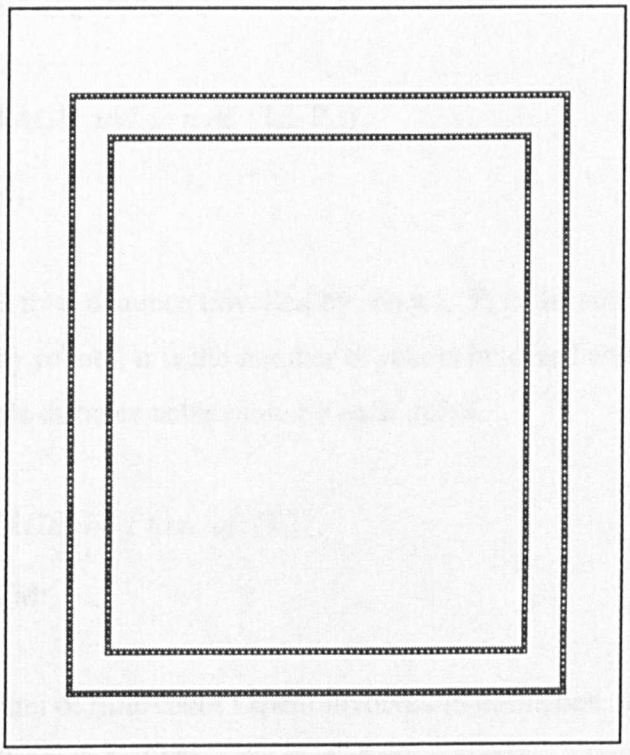


Figure 8.2 Test environment 2.

Motion is either forward or backward, and speed can be increased and decreased as required. The ability to detect and change track as well as the ability to detect collisions is also provided. Penalties are incurred for travelling fast and changing track. These penalties take the form of increased energy depletion and, in the case of track changing, constraint distance penalisation. The robots initially move forward and automatically negotiate all turning points on the track without additional instruction. Their progress around the track can only be halted by a collision with a static robot, a head-on collision with another robot, total depletion of their energy supply or the command to stop dead. Each robot is initially given 50 units of energy, which is reduced with every motion or attempted motion. In order to measure the performance of the programs two constraints were used:

1. The difference between the maximum possible distance travellable and the net distance travelled minus additional penalty for each track change,
2. The percentage of time spent involved in collisions.

These can be expressed as follows:

$$Ad = \{AVERAGE: i=1 \text{ to } n \text{ of } (Td_i - P_i)\},$$

$$c_1 = Md - Ad, \qquad \text{Equation 8.1}$$

where $\mathbf{Td_i}$ is the net total distance travelled by robot **i**, $\mathbf{P_i}$ is the sum of all distance penalties incurred by robot **i**, **n** is the number of robots involved and **Md** is the sum of maximum travellable distance achievable by each robot.

$$Ct = \{AVERAGE: i= 1 \text{ to } n \text{ of } (T_i)\},$$

$$c_2 = (Mt - Ct)/Mt. \qquad \text{Equation 8.2}$$

where $\mathbf{T_i}$ is the amount of time robot **i** spent involved in collisions, **n** is the number of robots involved in the task and **Mt** is the sum of the maximum amount of time for which each can run.

By minimising these constraints, the GP would be aiming to produce controllers which use the communication of both information and control as a method of maximising the distance travelled by the group of robots. The controllers to be produced utilise no ADFs and consist of a single main module in which all the communication and other actions take place.

## 8.3 FUNCTIONS AND TERMINALS

A list of the functions and terminals available to the evolutionary process is shown, along with their associated meaning in Table 8.1. The majority of the functions available are for communication purposes, with additional Boolean functions to allow for the combining of information request and general function sequencing. Terminals for accessing robot identification are also provided. Boolean *TRUE* is set at 1000 and Boolean *FALSE* to -(3*n). Where n is the number of robots. A *NULL* value is returned by some operators the value of which is -1.

The action set of functions (with general name structure A*) was chosen so that they would offer a basic set of activities which are available to all robots and which separately or combined would allow for a diverse set of strategies to emerge.

The information request set of functions (with general name structure Q*) were selected to mirror the actions provided, such that each action would benefit from asking a question prior and relevant to its use. Some actions work best with a single request, others with structured multiple requests.

| FUNCTIONS & TERMINALS | DESCRIPTION |
|---|---|
| -i | This is constant robot id denoted by the '-' sign, where i is any cardinal value greater than 1. |
| MyRid | Returns the robots id. |
| CurRid | Returns the robot's current sequential id value. |
| Rid>> | This causes the robot's sequential id pointer to be advanced forward to the next one, if last id exceeded, then it is reset to the first id or if robot's id then skip to next one. Returns the value of the robot's sequential id before it was advanced. |
| <<Rid | This cause the robot's sequential id pointer to be moved backward to the previous one, if it goes past the first id, then it is set to the last id or if robots id then skip to next one. Returns the value of the robot's sequential id before it was moved. |
| IFGE **a b c d** | If **a** is greater or equal to **b** then return **c** otherwise return **d**. |
| IFEq **a b c d** | If **a** is equal to **b** then return **c** otherwise return **d**. |
| Max **a b** | Returns the larger of the values **a** and **b**. |
| Min **a b** | Returns the lowest of the values **a** and **b**. |
| & **a b** | Returns a Boolean value reflecting the outcome of a logical AND operation between **a** and **b**. If either of **a** and **b** are not Boolean values it returns *FALSE*. |
| \| **a b** | Returns a Boolean value reflecting the outcome of a logical OR operation between **a** and **b**. If either of **a** and **b** are not Boolean values it returns *FALSE*. |
| ^ **a b** | Returns a Boolean value reflecting the outcome of a logical XOR operation between **a** and **b**. If either of **a** and **b** are not Boolean values it returns *FALSE*. |
| ~ **a** | Returns a Boolean value reflecting the outcome of a logical negation on **a**. If **a** is not a Boolean value then *FALSE* is returned. |
| QLane **a** | Request the robot indicated by **a** to confirm if it is on the same track as the current robot. Returns *TRUE* if correct and *FALSE if not.* |
| QTouch **a** | Request the robot indicated by **a** to confirm if it is touching the current robot. Returns *TRUE* if correct and *FALSE if not.* |
| QFacing **a** | Request the robot indicated by **a** to confirm if it is facing in the opposite direction to as the current robot. Returns *TRUE* if correct and *FALSE if not.* |
| QSection **a** | Request the robot indicated by **a** to confirm if it's current track section number is the same as the current robot. Returns *TRUE* if correct and *FALSE if not.* |
| QMoving **a** | Request the robot indicated by **a** to confirm if it's moving in the same direction as the current robot. Returns *TRUE* if correct and *FALSE* if not. |
| QStill **a** | Request the robot indicated by **a** to confirm if it is in stop mode. Returns *TRUE* if correct and *FALSE* if not. |

| FUNCTIONS & TERMINALS | DESCRIPTION |
|---|---|
| QFront **a** | Request the robot indicated by **a** to confirm if it is in front of the current robot. This is determined by the difference between section numbers. Returns *TRUE* if correct and *FALSE if not.* |
| QMxSp **a** | Request the robot indicated by **a** to confirm if it is moving at its maximum speed. Returns *TRUE* if correct and *FALSE if not.* |
| AChLane **a** | Makes the robot indicated by **a** change to the other track if possible. Returns *NULL* value. |
| ARvTog **a** | Makes the robot indicated by **a** change its direction of motion. If moving forward then it moves backwards and if moving backwards it moves forward. Returns *NULL* value. |
| AStop **a** | Makes the robot indicated by **a** stop moving and enter stop mode. Returns *NULL* value. |
| AAccel **a** | Makes the robot indicated by **a** increase its speed to next available gear. There are three possible gears: slow, cruise and fast. The initial gear is always set to cruise. Returns *NULL* value. |
| ADecel **a** | Makes the robot indicated by **a** decrease its speed to next available gear. There are three possible gears: slow, cruise and fast. The initial gear is always set to cruise. Returns *NULL* value. |

Table 8.1 List of available functions and terminals and their uses.

## 8.4 EXPERIMENTS AND ANALYSIS

*Aim:* To establish the level to which communication can be used to control directly the behaviour of other robots in an environment.

Owing to the processing load required for this task, a total of six additional HP 700 workstations were used as part of the distributed evaluation system to speed up processing time. Three distinct runs were used to accumulate results. Each of these runs had a different set of initial random seed values for the random streams used by the GP engine and the simulator.

### 8.4.1 Results

The time taken to run each of the three runs to completion varied from 10 to 12 days. The reason behind these long simulation times was primarily a factor of the number of robots involved in the simulation and the minimum speed (due to the fact that the robots could alter their speed) at which they could move. The runtime of a simulation is directly proportional to the number of robots in it and the amount of initial energy they have. Since twice as many robots are used here in comparison to the other experiments, this would result in at worst a doubling of the run-time of the system. In addition, the minimum speed used here is half that used in other simulations. So if this minimum speed was constantly used it would result in a doubling of the run-time of the system. The amount of initial energy units determines the stop time for the simulation. This value was kept the same as in other experiments for consistency reasons. The distributed evaluation method was used to reduce this burden, for without it, the computational times would have been longer. The variability in total simulation time was mostly governed by the speed at which the robots selected to travel at, since all other factors governing the performance remained constant.

The average time taken to produce the first tolerable controller in these runs was three generations. The strategies of the initial controllers were very specific to a single test combination and as a result produced controllers whose typical performance was poor on average [452 65] and with an average best performance of [390 18]. The average optimisation duration was 263 generations. The best controller found had an average best performance of [21 14] and worst of [43 21].

The evolution of controllers took place in several stages. Although the start, end and duration of these stages varied and in some cases overlapped, four main stages could be identified. These were marked by the use and targeting of communication and are given below:

1. direct control only, group orientated
2. direct control only, a mix of individually specific and group

3. combination of information request and direct control, group orientated
4. combination of information request and direct control, a mix of individually specific and group

Although initially the population of controllers was random, after about three to seven generations, controllers utilising distinctive and relevant strategies started to emerge. This was the prologue to stage 1. Stage 1 saw the evolution of controllers, which used communication to alter directly the behaviour of the other robots in the environment. This early initial use was due primarily to the immediate fitness benefit that could be realised from its use. These controllers were very simple and employed the use of a single instruction, which forced the receiver to change track. By using such a strategy of continually changing track, the probability of colliding with others would be reduced, hence the robots could travel further. This constant track changing resulted in fitness penalties every time it occurred, but the penalties were not sufficient to dampen totally the effectiveness of the strategy. This, as well as the lack of any other equally effective alternative, allowed these controllers to dominate the population for a long time. In order to improve the performance of this class of controllers, two approaches were prominent, each of which sought to reduce the frequency with which the robots changed track. The first approach did this by producing long programs with single change track commands in them, the second by developing sub-populations of communicating robots which only sent the change track commands to their members. Another, less effective approach, was to communicate "slow down" to all robots. This latter approach achieved a slightly higher level of performance when certain members of the population were made to slow down. The improvements offered by the two prominent alternatives established them as templates for improved performance at this stage. Those controllers which used the first approach sought to find the optimal delay between the changing of tracks, which leads to the best performance of the two, whereas those which used the second approach tried to reduce the number of robots commanding an individual to change track. A selection of some of the controllers produced in this stage can be seen in Figures 8.3 to 8.8. Figure 8.3 shows a controller using the general change track strategy. This was one of the first meaningful solutions found so the performance of the controller was quite bad. The performance of the controllers shown in Figures 8.4 and

8.5 show controllers from each of the two prominent approaches. The controller in Figure 8.4 uses its length to define the rate of track change (approach 1) and the controller in Figure 8.5 uses varying number of advances of the group robot id counter along with a condition based on *MyRid* to define communication sub-populations. The controller in Figure 8.6 applies a similar approach, but here, the sub-populations are of different sizes. Figure 8.7 shows a controller using speed to improve the distance the robots travel. Figures 8.8 shows a controller, which communicates with the whole group of robots but in a different sequence to default sequence. This is achieved by utilising multiple sequence id advances but all in the same direction.

```
system[
    main(0,-1)11[
        (&
            -2
            (ALane (Rid>>))
        )
    ]_Main
]_Sys [320 0] ↔ [317 0]
```

Figure 8.3 Initial effective controller.

```
system[
    main(0,-1)28[
        (QMxSp
            (^
                (IFEq
                    (& (<<Rid) (CurRid))
                    (IFEq
                        (<<Rid)
                        (IFGE
                            (~ -2)
                            (~ (MyRid))
                            THEN
                             (ALane (CurRid))
                            ELSE
                             (IFGE (CurRid) (<<Rid)
                                THEN (CurRid)
                                ELSE (MyRid)
                             )
                        )
                          THEN -7
                          ELSE -5
                    )
                      THEN
                       (ATogBk
                           (AUp (Rid>>))
                       )
                      ELSE
                       (QStill (MyRid))
                )
                (QFront (Rid>>))
            )
        )
    ]_Main
]_Sys [70 10] ↔ [63 4]
```

Figure 8.4 Communication command controller using program length to improve performance.

```
system[
    main{0,-1}28[
        (|
            (IFGE (CurRid) (MyRid)
               THEN (- (Rid>>))
               ELSE (Max (Rid>>) (Rid>>))
             )
            (ALane (CurRid))
        )
     ]_Main
 ]_Sys  [128 12] ↔ [124 8]
```

Figure 8.5 Communication command controller using sub-population communication to improve performance.

```
system[
    main{0,-1}28[
        (^
            (QMxSp
                (ALane
                    (IFGE (MyRid) (Rid>>)
                       THEN (<<Rid)
                       ELSE (CurRid)
                     )
                 )
             )
            (QStill
                (ATogBk
                    (QLane (<<Rid))
                 )
             )
        )
     ]_Main
 ]_Sys  [105 3] ↔ [101 11]
```

Figure 8.6 Controller using varying sized communication sub-populations.

```
system[
    main{0,-1}28[
        (~
            (Max
                (Rid>>)
                (ADown (CurRid))
             )
        )
     ]_Main
 ]_Sys  [304 43] ↔ [296 59]
```

Figure 8.7 Controller applying alternate strategy using slow down command.

```
system[
    main{0,-1}28[
        (&
            (QMxSp
                (ALane
                    (Max (<<Rid) (CurRid))
                 )
             )
            (QStill
                (ATogBk
                    (QLane (<<Rid))
                 )
             )
        )
     ]_Main
 ]_Sys  [130 4] ↔ [119 6]
```

Figure 8.8 Controller using non-standard communication sequence.

Stage 2 of controller evolution saw the use of command and execution targeting at individuals. Here, the controllers started to target specific commands at individual robots as well as allowing the execution of individual specific code fragments. The introduction of this explicit one-to-one communication between robots worked in parallel or in conjunction with the established group-orientated communication. As well as the existing method of defining communication subsets, another could now be adopted. Here, the controllers only utilise a small set of explicit robot identifications, which in conjunction with the use of individual specific code fragments allowed for comparable subsets. Further, the group orientated sub-set approach was extended to allow an alternate way of producing one-to-one communication. This was a very compact form, unlike that of the explicit ids. The main strategy remained that of changing tracks but now other commands were used in conjunction with it. These additional commands could be targeted at specific individuals, groups or sub-groups. This targeting of commands, and the ability to specify code fragment conditions allowed for different rates of track changing between the robots. This led to an improved level of performance of the controllers, which in turn helped increase the number of conditional dependent controllers. The conditions used by these controllers were all related in some way to the robot's identification value. Figures 8.9 to 8.15 show some of the controllers produced during this stage. The controller in Figure 8.9 shows a controller using individualised communication. Here, two of the robots do not receive any communication at all, resulting in their behaviour being unchanged. The use of totally explicit individualised communication requires that all the activities for all the robots should be stated. Consequently, long programs are required. Further, due to the lack of conditional execution, the rate of track changing is high leading in turn to relatively poor performance. The controller in Figure 8.10 shows a controller, which also uses individualised communication, but this time with a degree of conditional execution. Although not all the robots receive communication, the performance is better than that of Figure 8.9, which is due to the fact that each of the robots that receive communication only receive it from at most two robots. This reduces the change track rate for the robots, which is in contrast to the controller in Figure 8.9 where they receive it from all of the robots in the population including themselves, which in turn leads to a

high change track rate. Figure 8.11 shows a controller exhibiting the use of both individualised and group communication. The controller splits the robots into two sections, based on their id value. One section solely communicates with a single individual (those robots whose id is less than -5) and the other section alternates between individualised and group-orientated communication. Figure 8.12 shows a controller, which uses group-orientated communication operators to achieve solely individualised communication. This is achieved by balancing the number of forward and backward sequential robot id advances executed in the controller. The controller in Figure 8.13 uses conditional code fragments to produce different track changing rates for each robot. It achieves this by creating varying length routes through the controller, dependent on robot id. By varying the time it takes to pass through the controller, the frequency with which the robots send their individualised communication varies. Figure 8.14 shows a controller, which omits sending communication to single robots using the group communication. The use of multiple communication commands is exhibited by the controller in Figure 8.15. Here, the stop and speed up commands are used in conjunction with the main strategy. These additional commands are used sequentially to stop and start robots. As with this controller and many of the other multiple command controllers, as well as alternate strategies to the main one, the additional actions they generate are not very effective, because these actions happen in a haphazard fashion which lacks any particular relevance.

```
system[
    main{0,-1}28[
        (ATogBk
            (ALane
                (Max
                    (|
                        (ALane -2)
                        (Max
                            (ALane
                                (Min -7 (MyRid))
                            )
                            (^
                                (<<Rid)
                                (ALane -3)
                            )
                        )
                    )
                    -4
                )
            )
        )
    ]_Main
]_Sys  [361_17] ↔ [346_20]
```

Figure 8.9 Controller using explicit individualized communication.

```
system[
    main{0,-1}28[
        (^
            (IFEq (CurRid) -4
               THEN
                (ALane (CurRid))
               ELSE
                (IFEq
                    (Max
                        (MyRid)
                        (~ -6)
                    )
                   -7
                      THEN
                       (ALane -2)
                      ELSE
                       (QStill
                           (IFGE -2 (MyRid)
                              THEN
                               (ALane (MyRid))
                              ELSE
                               (ALane (MyRid))
                           )
                       )
                )
            )
            (~
                (QMxSp (CurRid))
            )
        )
    ]_Main
]_Sys  [149 20] ↔ [140 18]
```

Figure 8.10 Controller exhibiting heterogeneous functionality and individualized communication.

```
system[
    main{0,-1}28[
        (IFGE (MyRid) -5
           THEN
            (IFGE (CurRid) (MyRid)
               THEN
                (ALane (Rid>>))
               ELSE
                (QStill
                    (Max
                        (Rid>>)
                        (|
                            (ALane (MyRid))
                            (Rid>>)
                        )
                    )
                )
            )
           ELSE
            (Min
                (ALane (CurRid))
                (QTouch (CurRid))
            )
        )
    ]_Main
]_Sys  [140 13] ↔ [136 10]
```

Figure 8.11 Controller utilizing both group and individualized communication.

```
system[
    main{0,-1}28[
          (^
              (QMxSp
                  (Min
                      (~ (Rid>>))
                      (ALane (CurRid))
                   )
               )
              (~
                  (~
                      (QFront
                          (QMoving
                              (QFront (<<Rid))
                           )
                       )
                   )
               )
          )
      ]_Main
 ]_Sys  [129 15] ↔ [123 9]
```

Figure 8.12 Controller using group based communication to produce 1:1 communication.

```
system[
    main{0,-1}28[
          (Max
              (|
                  (IFEq (CurRid) -4
                     THEN -5
                     ELSE
                      (IFEq
                          (Max
                              (MyRid)
                              (~ -6)
                           )
                          -7
                             THEN
                              (ALane
                                  (Min
                                      (QSection
                                          (~ -4)
                                       )
                                      (QMxSp (MyRid))
                                   )
                               )
                             ELSE
                              (QStill
                                  (IFGE -2 (MyRid)
                                     THEN
                                      (QMxSp -3)
                                     ELSE
                                      (ALane (MyRid))
                                   )
                               )
                       )
                   )
                  (QFront
                      (AStop
                          (Min
                              -6
                              (~
                                  (QStill (CurRid))
                               )
                           )
                       )
                   )
               )
              (ALane (CurRid))
          )
      ]_Main
 ]_Sys  [132 23] ↔ [125 12]
```

Figure 8.13 Controller applying different change track rates to robots.

```
system[
    main{0,-1}28[
          (^
              (ALane
                  (Min
                      (Rid>>)
                      (Min (Rid>>) (CurRid))
                   )
               )
              (~ (MyRid))
          )
      ]_Main
 ]_Sys  [150 9] ↔ [139 8]
```

Figure 8.14 Controller using group communication to omit communication to specific individuals.

```
system[
    main{0,-1}28[
          (Min
              (IFGE (<<Rid) -6
                 THEN
                  (&
                     (<<Rid)
                     (AUp (MyRid))
                   )
                 ELSE (Astop (CurRid))
               )
              (~
                  (~
                      (QStill
                           (Alane (CurRid))
                       )
                   )
               )
          )
      ]_Main
 ]_Sys  [265 11] ↔ [261 9]
```

Figure 8.15 Controller using multiple command combinations along with main strategy.

As the controllers produced in stage 2 were gradually approaching a performance ceiling, there was a need for additional decision making or performance enhancing abilities. This emerged in stage 3, the arrival took a considerable amount of time, which is primarily attributed to the dominance of the initial style of controllers. In this stage, the controllers integrate both information requests and control commands into their communications. In these controllers, only group-orientated communication was used. The integration of information request within controllers allowed for improved targeting of action commands, which enabled non-standard strategies to improve their performance envelope. Further, it allowed for the development of new strategies and the enhancement of others. By using information request, strategies such as those given below could be formed:

- toggle direction of motion if collision detected
- slow down if approaching another robot and speed up if in front of one
- accelerate until touching robot moving in same direction then slow down

Although all strategies improved somewhat with the use of information requests, the main strategy reaped the greatest benefit, confirming it as the only viable option. Because of this, the other command actions tended to be used in a supporting role with it. The information requests were used in the main strategy to define the situations under which the change track action can take place. This is an additional way of minimising the change track rate, which also led to shorter programs in general and the decline of the controllers whose rate of track changing is governed by their length. However, although the rate of track changing in these controllers was reduced, the amount of time spent involved in collisions increased. This is attributable to inappropriate conditions, excessively detailed conditions and low rate of track changing in the controllers. This high level of collisions reduced the performance of these controllers to slightly below that of the best single track changing controller, and as a result of this the rate at which these combined communication controllers entered the population was slowed down. The use of information requests also provided for the ability to achieve a continuously changing communication sequence as well as an additional way for defining sub-populations. Communication sub-populations are achieved by stating the properties individuals must have in common, for example moving or being on the same track. Continuously changing communication can be achieved by defining criteria, which cause either the direction of motion group sequential id to change or for multiple individuals to be skipped. A sample of some of the controllers produced during this stage can be seen in Figures 8.16 to 8.20. The controller shown in Figure 8.16 implements an alternate strategy to the main one. This strategy requires a robot to change the direction of motion of any robot colliding with it. This is an effective strategy, in that it keeps the robots moving, however it does not maximise the number of circuits of the environment the robots makes. Therefore, it is penalised by the fitness function. The use of information request allowed for the rate at which robots changed direction to be minimised leading to an improved level of performance, down from

around 450 to about 300. This improvement comes from the ability to communicate command actions to specific individuals at relevant times and avoid non-essential communication with others. Although the use of information request is beneficial, excessive and irrelevant use can lead to its use being detrimental, as can be seen in Figure 8.17. In this controller, information requests are mis-used in two ways, first by defining a condition on mutually exclusive events, second, and most significant, is the use of over elaborate conditions. This can be seen in the emboldened section of the controller code. The harshness and length of this condition reduces the performance of the controller significantly. If this emboldened section of code were removed, the fitness of the controller would improve from a range of [278 53] ↔ [270 50] to one of [89 8] ↔ [85 3]. Figure 8.18 shows a controller that adds a simple information request to a controller based on the main change track strategy. Here it uses information on collision status to decide when to tell other robots to change track. This information is also used to trigger the use of the slow down command, such that the robot-changing track is also told to "slow down". Figure 8.19 shows another controller based on the standard strategy, this however augments it with a more complex but compact information request structure. This does not lead to improved performance but does reduce the length of the program required for that level of performance. The controller in Figure 8.20 uses information request to alter the communication sequence. This it achieves by temporarily changing the direction of motion of the group sequential id index.

```
system[
    main{0,-1}28[
        (IFGE
            (QTouch (CurRid))
            -3
               THEN
                (IFGE (CurRid) (MyRid)
                   THEN
                    (^
                        (QFacing (CurRid))
                        (Rid>>)
                    )
                   ELSE
                    (ATogBk (Rid>>))
                )
               ELSE
                (QMoving
                    (& (MyRid) (Rid>>))
                )
        )
    ]_Main
]_Sys  [315 31] ↔ [308 38]
```

Figure 8.16 Controller using information request and communication sub-populations to produce improved alternate strategy.

```
system[
    main{0,-1}28[
        (IFGE
            (Max
                (QTouch
                    (IFEq
                        (QMoving (CurRid)) // can not be moving and stationary at once
                        (QStill (CurRid))
                           THEN (MyRid)
                           ELSE (CurRid)
                    )
                )
                (QLane (Rid>>))
            )
            (QTouch (CurRid))
               THEN
                (IFGE
                    (Min  // over elaborate condition which is also hard to satisfy
                        (QTouch (CurRid))
                        (Min
                            (QFacing (<<Rid))
                            (Min
                                (QSection (Rid>>))
                                (QFront (CurRid))
                            )
                        )
                    )
                    (QLane (MyRid))
                       THEN
                        (ALane (CurRid))
                       ELSE
                        (- (CurRid))
                )
               ELSE
                (AUp (Rid>>))
        )
    ]_Main
]_Sys  [278 53] ↔ [270 50] :: [89 8] ↔ [85 3]
```

Figure 8.17 Controller exhibiting detrimental use of information requests.

```
system[
    main{0,-1}28[
        (ADown
            (Min
                (QSection
                    (QMoving (<<Rid))
                )
                (IFGE (QFront (CurRid)) (MyRid)
                   THEN
                    (Min
                        (ALane (CurRid))
                        (<<Rid)
                    )
                   ELSE
                    (QStill -5)
                )
            )
        )
    ]_Main
]_Sys  [56 3] ↔ [53 5]
```

Figure 8.18 Simple information request incorporated within a multiple command action strategy.

```
system[
    main{0,-1}28[
        (ALane
            (Min
                (Rid>>)
                (Max
                    (|
                        (QStill (CurRid))
                        (QTouch (CurRid))
                    )
                    (Max
                        (&
                            (QTouch (CurRid))
                            (QMoving (CurRid))
                        )
                        (QTouch (CurRid))
                    )
                )
            )
        )
    ]_Main
]_Sys  [70 10] ↔ [67 12]
```

Figure 8.19 Compact use of multiple information request with main strategy.

```
system[
    main{0,-1}28[
        (^
            (QMxSp
                (ALane
                    (IFGE
                        (QLane (Max (Rid>>) (MyRid)))
                        (Rid>>)
                        THEN (<<Rid)
                        ELSE
                        (ALane
                            (Min
                                (QLane (~ (Rid>>)))
                                (CurRid)
                            )
                        )
                    )
                )
            )
            (QStill
                (ATogBk (ALane (<<Rid)))
            )
        )
    ]_Main
]_Sys  [126 3] ↔ [123 1]
```

Figure 8.20 Information request used to determine communication sequence as well as command action population.

The fourth stage always commenced in the last few generations of each run. The controllers produced here utilised both forms of communication (information request and commands) as well as both forms of robot identification (individual and group). This improved on the stage 3 controllers in that it allowed specific information to be elicited from certain robots. This information could then be used to determine what commands to communicate. This ability to individualise communication again allowed for specific targeting of commands to individual robots. This stage saw the development of most of the optimal controllers, their main strategy revolving around track changing

and determining the appropriate time for it based on information requests. The strategy of one of the most effective controllers was to command any other robot on the same track and sector as the robot, but moving in the opposite direction to it, to change track. This stage also saw the improvement in performance of most of the other controllers as well as a steady decline in their numbers. It also saw the general demise of long programs in favour of short, information-dependent ones. By using information about specific individuals, somewhat elaborate controllers tended to emerge. Again, the problem of over verbose conditions remained. Some of the controllers produced during this stage can be seen in Figures 8.21 to 8.25. Figure 8.21 shows a controller in which robot *5* has its own separate behaviour from the rest of the group. This is to stop if it is touching someone otherwise, it should carry on moving. A clever use was made here of the *AUp* operator in conjunction with the *QStill* operator to ensure that the robot was only told to speed up if it was in stop mode. In addition, a different information request order was used by the robot. The rest of the robots applied a strategy of changing track if a collision occurs. This is similar to the strategy employed by the controller in Figure 8.22. This controller employs the use of the minimal amount of information request appropriate to the strategy. It uses a single request to see if a collision has taken place, which, if it has, causes a change track command to be issued. It uses a peculiar command targeting method. Instead of commanding the individual colliding with it to change tracks, it commands the previous individual in the robot index sequence to change track. This strategy, though effective, is dependent on the order in which collisions take place, as well as who is involved in them. Figure 8.23 shows a very effective controller using individualised command action communication and group orientated information request communication. This controller again uses the main strategy, but this time its use is governed by two information requests. These are, "are you in the same lane?" and "are you facing in the opposite direction?" The effectiveness of this strategy is due to the fact that it quickly attempts to get all the robots moving in the same direction onto the same track. The controller in Figure 8.24 applies a similar strategy but its communications are all group orientated. Figure 8.25 shows a controller using individualised information request to determine which of its two possible strategies to employ. Robots *4*, *2* and *6* are quizzed to see if they are touching the current robot. If any of them are then the current robot is commanded to change track, if

their not then the current robot uses a strategy similar to that of the controller in Figure 8.24 to determine who to command the change track.

```
system[
    main{0,-1}28[
        (IFEq (MyRid) -5
           THEN
            (IFGE
                (QTouch (Rid>>))
                -4
                   THEN
                    (AStop (MyRid))
                   ELSE
                    (AUp
                        (Min
                            (MyRid)
                            (QStill (MyRid))
                        )
                    )
            )
           ELSE
            (ALane
                (Min
                    (<<Rid)
                    (QTouch (CurRid))
                )
            )
        )
    ]_Main
]_Sys  [64 8] ↔ [60 11]
```

Figure 8.21 Controller featuring heterogeneous functionality with varying information requests.

```
system[
    main{0,-1}28[
        (ALane
            (Min
                (Rid>>)
                (QTouch (CurRid))
            )
        )
    ]_Main
]_Sys  [30 1] ↔ [21 5]
```

Figure 8.22 Compact controller using minimal information request for command action employed and exhibiting collision order dependence.

```
system[
    main{0,-1}28[
        (&
            (ALane
                (Min
                    (MyRid)
                    (Min
                        (QLane (CurRid))
                        (QFacing (CurRid))
                    )
                )
            )
            (<<Rid)
        )
    ]_Main
]_Sys  [15 1] ↔ [9 0]
```

Figure 8.23 High performing controller using 1:1 command communication and multiple group orientated information requests.

```
system[
    main{0,-1}28[
         (Min
              (Rid>>)
              (IFGE
                   (Min
                        (QLane (CurRid))
                        (QFacing (CurRid))
                    )
                   (QLane (MyRid))
                      THEN
                       (ALane (CurRid))
                      ELSE
                       (QFront (CurRid))
               )
          )
      ]_Main
 ]_Sys  [14 9] ↔ [8 0]
```

Figure 8.24 High performing controller based on group orientated communication and utilizing multiple information requests.

```
system[
    main{0,-1}28[
         (IFGE
              (Max
                   (QTouch -4)
                   (|
                        (QTouch -2)
                        (QTouch -6)
                    )
               )
              (QLane (MyRid))
                 THEN
                  (ALane (CurRid))
                 ELSE
                  (ALane
                       (Min
                            (Min
                                 (QFacing (CurRid))
                                 (QLane (CurRid))
                             )
                            (<<Rid)
                        )
                   )
          )
      ]_Main
 ]_Sys  [53 1] ↔ [46 7]
```

Figure 8.25 Controller utilizing individualized information requests to determine which of two possible strategies to apply.

### 8.4.2 Discussion of results

The evolution of controllers, although slow and prone in some cases to over dominance by some controllers, proved to be able to make use of communication in both its roles (a summary of the controllers presented in this chapter can be seen in Table 8.2). The most readily exploitable and beneficial form was for direct control. Further more the use of information elicitation alone could not achieve any meaningful solution to the problem. However, when used in conjunction with command-based communication, it helped to

extend the repertoire of the controllers. The lack of usefulness offered by information elicitation alone was due to the fact that the actions of the robots could not be altered. This altering of actions was the essential factor in improving the performance of the controllers. Consequently, the use of information request calls diminish in the population until a suitable command-based controller is available for its use. The structure of such a controller must be correct, so must be the positioning and relevancy of the request and individual from which the information is requested. Such controllers take time to produce or may not even be produced due to the pressures of purely command-based controllers. This need for the existence of compatible controllers leads to the prolonged time before information based communications offer benefits to evolutionary process and can establish a significant presence in the population. Another factor in this delay is the instant benefit offered by the use of the change track command. This also limited the development of alternate strategies. This is mainly due to two factors, firstly a combination of functions specified in the correct order are required for the strategy and secondly information is required to govern when a function or functions should be used. This leads to the initial bias of the population in favour of the change track based strategy. Premature convergence of the population to a simple change track approach is possible if the information request individuals are crowded out of the population. For the most effective strategies to be developed required the combination of the right command and information pairings as well as relevant program structure and length.

The range of actions available to the controllers limits the number of effective strategies that can be produced, many of the commands offered here are context sensitive. This means that for their true benefit to be realised they must be used under the appropriate circumstance. This limited the amount of possible distinctive solutions but it did not restrict the number and type of variants based on the dominant strategy from emerging.

Although a homogeneous system was set up, the evolutionary process was able to produced controllers with a degree of heterogeneity about them. They allowed individual robots to perform specific instructions. They also allowed for individualised

communication between robots as well as groups. This allowed for the creation of controllers with various forms of communication topologies. These were either static or dynamic in structure and membership. Membership of communication topologies could change depending on the type of communication being used. Also, due to the use of sub-populations there could exist controllers, which used numerous disjoint topologies. Sub-populations of the robot population could communicate with each other. This type of topology was only of benefit as long as the system as a whole had access to all relevant information about individuals in the environment or the degree of interaction between the individuals was limited to their sub-populations. The degree of individualism allowed by these controllers proved in must cases not to be detrimental. Such controllers were found to be outperformed by those controllers which applied a largely collective communication strategy.

| CHAPTER 8 FIGURE REFERENCE | SUMMARY OF CONTROLLER | BEST FITNESS | | WORST FITNESS | |
|---|---|---|---|---|---|
| | | constraint 1 | constraint 2 | constraint 1 | constraint 2 |
| 3 | Initial effective controller using only single control command. | 317 | 0 | 320 | 0 |
| 4 | Program length used to improve performance. | 63 | 4 | 70 | 10 |
| 5 | Communication sub-groups used. | 124 | 8 | 128 | 12 |
| 6 | Varied sized communication sub-groups. | 101 | 11 | 105 | 3 |
| 7 | Non standard control command used. | 296 | 59 | 304 | 43 |
| 8 | Alternate communication order from normal used. | 119 | 6 | 130 | 4 |

| CHAPTER 8 FIGURE REFERENCE | SUMMARY OF CONTROLLER | BEST FITNESS | | WORST FITNESS | |
|---|---|---|---|---|---|
| | | constraint 1 | constraint 2 | constraint 1 | constraint 2 |
| 9 | Single control command combined with explicit individualised communication. | 346 | 20 | 361 | 17 |
| 10 | Heterogeneous functionality and individualised communication. | 140 | 18 | 149 | 20 |
| 11 | Group and individualised communication used. | 136 | 10 | 140 | 13 |
| 12 | Individualised communication using group communication operators. | 123 | 9 | 129 | 15 |
| 13 | Different change track rates used for each robot. | 125 | 12 | 132 | 23 |
| 14 | Communication Sub-groups created by omitting individuals from group communication. | 139 | 8 | 150 | 9 |
| 15 | Controller using multiple control commands. | 261 | 9 | 265 | 11 |
| 16 | Both forms of communication utilised to improve alternate strategy. | 308 | 38 | 315 | 31 |
| 17 | Detrimental use of information requests. | 270 | 50 | 278 | 53 |
| 18 | Simple and effective information request used in multiple control command strategy. | 53 | 5 | 56 | 3 |

| CHAPTER 8 FIGURE REFERENCE | SUMMARY OF CONTROLLER | BEST FITNESS | | WORST FITNESS | |
|---|---|---|---|---|---|
| | | constraint 1 | constraint 2 | constraint 1 | constraint 2 |
| 19 | Compact use of multiple information requests. | 67 | 12 | 70 | 10 |
| 20 | Information request used to determine which control commands to use. | 123 | 1 | 126 | 3 |
| 21 | Heterogeneous functionality and varying information requests. | 60 | 11 | 64 | 8 |
| 22 | Compact uses minimal information requests. | 21 | 5 | 30 | 1 |
| 23 | Uses 1:1 control command communication and group communication for information requests. | 9 | 0 | 15 | 1 |
| 24 | Uses group communication for both control commands and information requests. | 8 | 0 | 14 | 9 |
| 25 | Uses individualised information requests to determine which of two possible strategies to apply. | 46 | 7 | 53 | 1 |

Table 8.2 Summary table showing the performance of the evolved controllers reported in this chapter.

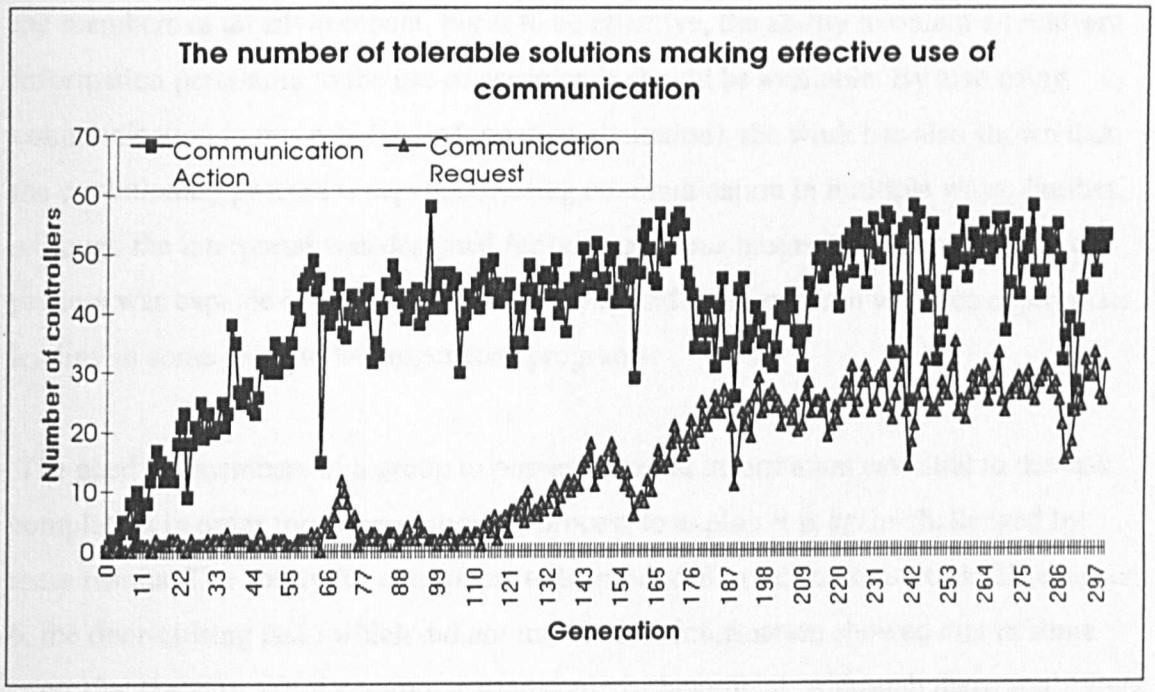
Graph 8.1 shows the rate at which controllers were found, which made use of communication, in either a command or an elicitation role. It can be seen that the trend towards using communication for command actions is generally upwards and starts very early on, which ties in with the early discovery of simple change-track controllers. The

randomness of the graph is attributable to the dynamic nature of the testing system. Since what is being displayed here is dependent on the content of the tolerable file, those occasions in which difficult tests occur will correspond to a marked and temporary drop in performance or indeed those tests for which the majority of the population are not optimal will also result in the same effect. A similar case exists for simple tests, where there will be a marked temporary upward jump in the number. The case for the use of information request shows its use also starts early, but it is in very small amounts and doesn't really catch on until much later on in the run. This early usage of information request can also be seen by spikes in the request graph. There are two possible reasons for the failure to catch on. Firstly, its application to poor or ineffective strategies, and secondly a highly disruptive reproduction process. The first case is most suitable to the continuing existence of low levels of controllers, the second to sharp jumps in the graph which do not correspond to jumps in the command action graph. The disruptive nature of the evolutionary process makes it hard for collections of genetic material to stay together, nevertheless this disruptiveness is needed to ensure a good level of exploration of the program space is achieved by the GP.

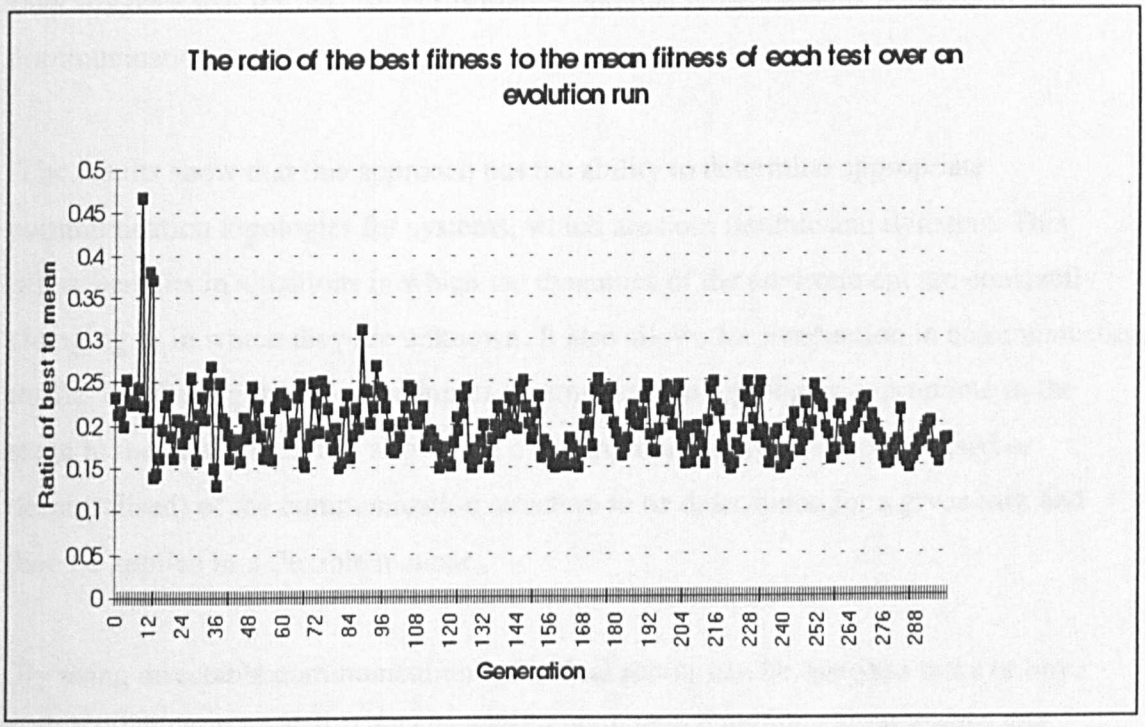
A point to be noted is that the information request line never crosses the command action line in any the graphs produced from any of the runs. If this were to happen, it would signify that solutions to the task are possible without the need for command actions.

Graph 8.2 shows the ratio of the best-to-mean fitness for each generation of a run. For this purpose, the fitness was reduced to a scalar value by summing all the vectors. This use of these ratios allows for relative comparisons between tests as opposed to the absolute comparison that would be achieved if respective individual values were considered. This is very pertinent here, since the tests are changing every generation. By making use of the ratio, a large degree of the noise can be eliminated. Although no general upward or downward trend is visible in this graph, it is notable that the ratios seem to stay within a fixed range (0.15 to 0.25). Two possible reasons can account for this, first, regardless of the difficulty or ease of the test, the performance in each test shows a fairly consistent ratio between the ratio pairings, secondly, the existence of a

balanced increase and decrease of ratio pairs. For example a 10% improvement in the best fitness is matched by a similar, possibly timelagged, improvement in the mean fitness.




Graph 8.1 Tolerable solutions making effective use of communication.




Graph 8.2 Typical change in the ratio of best to average fitness over a run.

## 8.5 SUMMARY

Controllers can be evolved which use communication as a method to control directly the members of an environment. For it to be effective, the ability to obtain all relevant information pertaining to the use of commands should be available. By also using communication in this role (i.e. information elicitation), the work has also shown that the evolutionary process is capable of using communication in multiple ways. Further, although the interpreter was designed for homogeneous programs, the evolutionary process was capable of exploiting commonality and individualism when its appropriate, leading in some cases to heterogeneous programs.

The need for members of a group to possess internal information essential to the task completion in order for the evolutionary process to exploit it is again challenged by these results. The ability for controllers to be produced in this task and task III (chapter 6, the door-closing task) which did not use internal information showed that in some cases that the very act of communication could be beneficial. Although these controllers benefit from the act of communication, their performance is not very optimal, which may suggests that for optimal performance, internal information is required in the communications.

The results show that this approach has the ability to determine appropriate communication topologies for systems, which are both flexible and dynamic. This offers benefits in situations in which the dynamics of the environment are constantly changing or in which they are unknown. It also allows for a reduction in communication traffic, by altering the membership of communication topologies appropriate to the stage in the task. Further, it allows the most appropriate style (i.e. centralised or decentralised) of the communication structure to be determined for a given task and then be applied in a flexible manner.

By using directable communication individual robots can be assigned tasks or have their behaviour restricted. This can also be applied to communication groups and

communications between these groups take place between group leaders or individuals in the group. This allows a task to be decomposed and distributed between groups of robots in the system.

# Chapter 9

# CONCLUSION

## 9.1 OVERVIEW

This thesis has addressed the following question:

> "Can evolutionary approaches applied to robotic problems help us to develop robust autonomous robots".

Two areas were investigated to answer this question. Firstly, the case of path planning for a mobile robot in a static environment was considered, using Genetic Algorithms. Secondly, communication between groups of robots and more specifically the evolution of communication control software using Genetic Programming was investigated.

The fundamentals of genetic algorithms and genetic programming were introduced in chapter 2. This provided the theoretical foundation upon which the work of this thesis was built.

A review of the literature was presented in chapter 3. Of current path planning techniques, the configuration space and the potential field methods were found to be the most prominent. The configuration space method was found to suffer from the following shortcomings:

- High computational demand for mapping between the real world and configuration space.
- Long search time for paths.
- Approximations errors, which lead to the failure to find paths when they exist.

Potential field methods were found to exhibit the following shortcomings:

- The existence of multiple local minima in the potential field, resulting in null driving forces and entrapment of the robot.
- The inability to guarantee convergence to goal in the presence of local minima.
- Difficulty in generating globally optimal paths.

With these shortcomings in mind, it was decided to investigate the application of genetic algorithms to the mobile robot path planning problem with respect to the following points:-

- How best to represent the environment.
- How best to represent the path.
- To identify the key features of the path planning process which can serve to guide the evolutionary process towards effective robust solutions.

In addition, a review of the work relating to teams of robots and communication between the robots in particular was presented. It was highlighted that no work addressing the question of what information to communicate and when to communicate it had been reported. This question was to be addressed in this using genetic programming.

Chapter 4 presented a series of experiments which investigated the genetic algorithm as a robot path planner. The following key points were identified:

A path representation using relative motions was shown to be appropriate.
Multiple constraints are required to satisfactorily derive paths. A prioritised multiple constraint system was shown to be achievable using a vector fitness representation.

The nature of environments and the scenarios that present the greatest problems to a GA path planner were highlighted. Various intelligent operators which allow for this problem to be successfully overcome were introduced and demonstrated.
Robustness problems can be eliminated using avoidance contours.
The performance of the GA path planner is comparable to that of existing techniques but its computational demand is greater than that of potential field approach but less than configuration space.

Chapter 5 introduced the experimental environment developed to investigate communication between multiple robots.

Chapter 6 presented the first experiments, performed with the aim to evolve communicating robot controllers. The task given was for a number of robots to meet up by communicating with each other. Various forms of communication were investigated. The results can be summarised as follows:-

- The evolutionary process can benefit from the presence of communicated information.
- The content of the information communicated is dependent on the task being performed.
- The range of communication is important.

Chapter 7 built on the results of chapter 6 and investigated information content and when to communicate. The results can be summarised as follows:-

- The evolutionary process has the ability to decide when and what type of information to communicate.
- Continuous transmission was shown to be the best policy.

Chapter 8 presented experiments derived to investigate the ability of the evolutionary process to utilise communication as a way of controlling the actions of other robots in the environment. The results can be summarised as follows:

- Controllers can be evolved which use communication to directly control other robots in the environment.
- The evolutionary process is capable of using communication in many ways.
- The approach has the ability to determine appropriate communication topologies between teams of robots.

## 9.2 FURTHER WORK

The work presented here can be extended in several ways:

- The incremental growth theme could be examined, more closely addressing issues of self-improvement and maintenance of robots. Also, ways to produce systems which combine disjoint or competing competencies to produce more adaptable robot systems may be investigated.
- The theme of evolvable communication between robots can be further investigated to see if a group of mobile robots can evolve a minimal and consistent or workable language amongst themselves to aid in task completion. Also the evolution of heterogeneous controllers which divide elements of the task up between all environment members.

## 9.3 IN CONCLUSION

This thesis has identified the limitations of current mobile robot path planners. The use of genetic algorithms as an alternative method has been investigated and shown to be a viable and effective alternative. The question of communication between teams of

robots has also been addressed. Here genetic programming has been shown to be a suitable tool for evolving controllers that can decide when and what to communicate as well as to whom. As a result of these investigations, the question this thesis addressed has been answered. In particular, a thorough the investigation of GA path planning has been presented making an original contribution in the field of mobile robot path planning. All the pertinent questions have been answered resulting in a thorough methodology that is generally applicable. Further, the question of communicating had been answered. It has been shown that communicating improves task efficiency but more importantly that the question of when, what and to whom to communicate to can be automated using a GP, removing this burden from the designer. This represents an original contribution to the field of mobile robotics. Further, it represents one of the most demanding task placed on the GP reported in the literature to date, in terms of the size of the function and terminal set used.

# References


1. J.Kosecka and H.I Christensen, 1995, Experiments in behavior composition., *Proceedings of the 3rd International Symposium on Intelligent Robotic Systems*, pp 129-139.

2. R.M Rylatt, C.A Czarnecki and T.W Routen, 1996, Learning to behave: an investigation of connectionist approaches to behaviour based control in autonomous agents., *Proceedings of IEEE 8th mediterranean Electrotechnical Conference*, pp 211-214.

3. R.A Brooks, 1986, A robust layered control system for a mobile robot., *IEEE Journal of Robotics and Automation RA-2*, pp 14-23.

4. C.Harston, 1990, Application of neural networks to robotics., *The Handbook of Neural Computing Applications*, Academic Press, pp 381-391.

5. G.Krishnaswamy, M.H Ang JR and G.B Andeen, 1991, Structured neural-network approach to robot motion and control., *Proceedings of the International Joint Conference on Neural Networks*, pp 1059-1066.

6. I.Harvey, P.Husbands and D.Cliff, 1992, Issues in evolutionary robotics., *Cognitive Science Research Paper*, CSRP 219 University of Sussex.

7. M.J.Mataric, 1995, Issues and approaches in the design of collective autonomous agents., *Robotics and Autonomous Systems*, Vol. 16, No. 2-4, pp 321-331 .

8. R.A Brooks, 1990, The behavior language; user's guide., *Technical Report*, A.I.Memo 1227, MIT AI Lab.

9. M.Mataric and D.Cliff, 1996, Challenges in evolving controllers for physical robots., *Robotics and Autonomous Systems*, Vol. 6, pp 67-83.

10. T.Baeck and H.P Schwefel, 1993, An overview of evolutionary algorithms for parameter optimisation., *Evolutionary Computing*, pp 1-24.

11. W.B Langdon and A.Quershi, 1995, Genetic programming -computers using 'natural selection' to generate programs., *Research Note RN/95/76*, University College London, "ftp://cs.ucl.ac.uk/genetic/papers/surveyRN76.ps".

12. D.E Goldberg, 1989, Genetic algorithms in search optimisation and machine learning., *Addison-Wesley publishing company.*

13. J.R Koza, 1992, Genetic programming: On the programming of computers by natural selection., *MIT press*, pp 192.

14. L.Davis, 1991, Handbook of genetic algorithms., *Van Nostrand Reinhold.*

15. L.J Eshelman and J.D Schaffer, 1991, Preventing premature convergence in genetic algorithms by preventing incest., *Proceedings of the 4th International Conference on Genetic Algorithms*, Morgan Kaufman, pp 114-122.

16. Z.Michalewicz and C.Z Janikow, 1991, Handling constraints in genetic algorithms., *Proceedings of the 4th International Conference on Genetic Algorithms*, Morgan Kaufman, pp 151-157.

17. J.R Levenick, 1991, Inserting introns improves genetic algorithms success rate: taking a cue from biology., *Proceedings of the 4th International Conference on Genetic Algorithms*, Morgan Kaufman, pp 123-127.

18. P.Husbands and F.Mill, 1991, Simulated co-evolution as the mechanism for emergent planning and scheduling., *Proceedings of the 4th International Conference on Genetic Algorithms*, Morgan Kaufman, pp 264-270.

19. D.E Goldberg, K.Deb and B.Korb, 1991, Don't worry be messy., *Proceedings of the 4th International Conference on Genetic Algorithms*, Morgan Kaufman, pp 24-30.

20. J.D Schaffer, 1985, Multiple objective optimisation with vector evaluated algorithms., *In Proceedings of the First International Conference on Genetic Algorithms and their Applications*, Lawrence Erlbaum Associates: Hillsdale, New Jersey, pp 93-101.

21. J.R Koza, 1992, Genetic programming: On the programming of computers by natural selection., *MIT press.*

22. J.R Koza, 1994, Genetic programming II: Automatic discovery of reusable programs., *MIT press.*

23. D.J Montana, 1994, Strongly typed genetic programming., *BBN Technical Report* #7866.

24. M.J Keith and M.C.Martin, 1994, Genetic programming in C++: Implementation issues., *In Advances in genetic programming, chapter 13*, MIT press, pp 285-310.

25. A.Teller, 1994, Genetic programming, indexed memory., the halting problem and other curiosities., *Proceedings of the 7th Annual Artificial Intelligence Research Symposium*, pp 270-274.

26. W.R Pringle, 1995, ESP: Evolutionary structured programming., *Technical report*, Penn State University, "http://www.gv.psu.edu/personal/wrp103/esp.ps".

27. K.E Kinnear Jr, 1993, Generality and difficulty in genetic programming: evolving a sort., *Proceedings of the 5th International Conference on Genetic Algorithms*, Morgan Kaufmann, pp 287-294.

28. C.W Reynolds, 1994, Evolution of corridor following behaviour in a noisy world., *From animals to animats (the proceedings of simulation of adaptive behaviour)*, MIT press, pp 402-410.

29. C.W Reynolds, 1993, An evolved vision based behavioural model of obstacle avoidance behaviour., *Artificial Life III*, Addison-Wesley, pp 327-346.

30. K.E Kinnear Jr, 1994, Alternatives in automatic function definition: Acomparison of performance., *In Advances in Genetic Programming*, MIT Press, Chapter 6, pp 119-142.

31. G.F Spencer, 1994, Automatic generation of programs for crawling and walking., *In Advances in Genetic programming*, MIT Press, Chapter 15, pp 335-353.

32. O.Khatib, 1985, Real-time obstacle avoidance for manipulators and mobile robots., *Proceedings of the IEEE International Conference on Robotics and Automation*, pp 500-505.

33. J.Kim and P.K Khosla, 1992, Real-time obstacle avoidance using harmonic potential functions, *IEEE Transactions on Robotics and Automation*, Vol. 8 No. 2 pp 77-86.

34. J.Barraquand and J.-C Latombe, 1991, Robot motion planning: A distributed representation approach., *International Journal of Robotics Research*, vol. 10, No.6, pp 628-649.

35. T.Ikegami and S.Ozono, 1992, A new path planning method for mobile robots applicable to an arbitrary environment., *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Raleigh, NC, pp 453-460.

36. Y.Koren and J.Borenstein, 1991, potential field methods and their inherent limitations for mobile robot navigation., *Proceedings of the IEEE International Conference on Robotics and Automation*, Saracamento, pp 1398-1404.

37. P.Khosla and R.Volpe, 1988, Superquadratic artificial potential for obstacle avoidance and approach., *Proceedings of the IEEE International Conference on Robotics and Automation*, Philadelphia, pp 1778-1784.

38. M.D.Adams, H.Hu and P.J. Probert, 1990, Towards a real-time architecture for obstacle avoidance and path planning in mobile robots., *Proceedings of the IEEE International Conference on Robotics and Automation*, Cincinnati, pp. 584-589.

39. R.C.Arkin, 1989, Motor schema-based mobile robot navigation., *International Journal of Robotics Research*, Vol. 8, No.4, pp 92-112.

40. C.W.Warren, J.C.Danos and B.W.Mooring, 1989, An approach to manipulator path planning., *International journal of robotics research*, Vol. 8 No. 5 pp 87-95.

41. A.A.Masoud and M.M.Bayoumi, 1993, Robot navigation using the vector potential approach., *IEEE International Conference on Robotics and Automation*, Vol. 3, pp 805-811.

42. E.Rimon and D.E.Koditschek, 1992, Exact robot navigation using potential fields., *IEEE Transactions on Robotics and Automation*, Vol. 8, No. 5 pp 501-518.

43. G.k.Schmidt and K.Azarm, 1992, Mobile robot navigation in a dynamic world using an unsteady diffusion equation strategy., *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Raleigh, NC, pp 642-647.

44. S.Akishita, S.Kawamura and K.Hayashi, 1990, New navigation function utilising hydrodynamic potential for mobile robot., *IEEE International Workshop on Intelligent Motion Control*, Istanbul, pp 413-417.

45. S.Ratering and M.Gini, 1993, Robot navigation in a known environment with unknown moving obstacles., *IEEE International Conference on Robotics and Automation*, Vol. 3, pp 25-30.

46. S.Akishita, T.Hisanobu and S.Kawamura, 1993, Fast path planning available for moving obstacle avoidance by use of laplace potential., *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Yokohama, Japan, pp 673-678.

47. J.Borenstein and Y.Koren, 1989, Real-time obstacle avoidance for fast mobile robots., *IEEE Transactions on systems, Man. and Cybernetics*, Vol. 19, No. 5, pp 1179-1187.

48. A.Manz, R.Liscano and D.Green, 1991, A comparison of realtime obstacle avoidance methods for mobile robots., *The 2nd International Symposium Experimental Robotics II*, Toulouse, France, Springer-Verlag, pp 299-316.

49. T.Lozano-Perez and M.A.Wesley, 1979, An algorithm for planning collision-free paths among polyhedral obstacles., *Communication of the ACM.*, Vol. 22, No. 10, pp 560-570.

50. T.Lozano-Perez, 1983, Spatial planning: A configuration space approach., *IEEE Transactions on Computers*, Vol. C-32, No. 2, pp 108-120.

51. T.Lozano-Perez, 1987, A simple motion-planning algorithm for general robot manipulators., *IEEE Journal of Robotics and Automation*, Vol. RA-3, No. 3, pp 224-238.

52. I.Hara and T.Nagata, 1991, Describing moving spherical obstacle in a configuration space., *IEEE International Workshop on Intelligent Robots and Systems*, pp 589-594.

53. S.K.Tso and K.P.Liu, 1993, A fast motion planner based on configuration space., *Proceedings of the IEEE Conference on Intelligent Robots and Systems*, pp 1401-1408.

54. J.T.Schwartz and M.Sharir, 1983, On the piano movers problem, the case of two dimensional rigid polygonal body moving amidst polygonal barriers., *Communications Pure Applied Mathematics.*, pp 345-398.

55. R.A.Brooks and T.Lozano-Perez, 1985, A subdivision algorithm in configuration space for findpath with rotation., *IEEE Transactions on Systems Man. and Cybernetics*, Vol. 15, No. 2, pp 224-233.

56. D.Zhu and J.C.Latombe, 1989, New heuristic algorithms for efficient hierarchical path planning., *IEEE Transactions on Robotics and Automation*, Vol. 7, No. 1, pp 9-20.

57. S.Quinlan and O.Kahtib, 1991, Towards real-time execution of motion tasks., *The 2nd International Symposium Experimental Robotics II*, Toulouse, France, Springer-Verlag, pp 241-254.

58. R.A.Brooks, 1983, Solving the findpath problem by good representation of free space., *IEEE Transactions on Systems Man. and Cybernetics*, Vol. 13, No. 3, pp 190-197.

59. Y-H Lui, S.Kuroda, T.Naniwa, H.Noborio and S,Arimoto, 1989, A practical algorithm for planning collision-free coordinated motion of multiple mobile robots., *IEEE International Conference on Robotics and Automation*, Vol. 8, pp 1427-1432.

60. H.Noborio, T.Naniwa and S.Arimoto, 1989, A feasible motion-planning algorithm for a mobile robot on a quadtree representation., *IEEE International Conference on Transactions and Automation*, Vol. 8, pp 327-332.

61. E.K.Wong and K.S.Fu, 1985, A hierarchical orthogonal space approach to three-dimensional path planning., *IEEE International Conference on Robotics and Automation*, pp 506-511.

62. K.Fujimura and H.Samet, 1989, A hierarchical strategy for path planning among moving obstacles., *IEEE Transactions on Robotics and Automation*, Vol. 5, pp 61-69.

63. A.O.Tukuta, 1991, Motion planning using binary space partitioning., *IEEE/RSJ International Workshop on Intelligent Robots and Systems IROS*, pp 86-90.

64. M.K.Habib and H.Asama, 1991, Efficient method to generate collision free paths for autonomous mobile robot based on new free space structuring approach., *IEEE/RSJ International Workshop on Intelligent Robots and Systems IROS*, pp 363-367.

65. K.Hughes, A.Tokuta and N.Ranganathan, 1992, trulla: an algorithm for path planning among weighted regions by localized propagations., *IEEE International Conference on Intelligent Robots and Systems*, pp 469-476.

66. A.Zelinsky, 1994, Using path transforms to guide the search for findpath in 2d., *International Journal of Robotics Research*, Vol. 13, No. 4, pp 315-325.

67. K.Fujimura and H.Samet, 1989, Time minimal paths among moving obstacles., *IEEE International Conference on Robotics and Automation*, pp 1110-1115.

68. C.L.Shih, T.T.Lee and W.A.Gruner, 1990, Motion planning with time-varying polyhedral obstacles based on graph search and mathematical programming., *IEEE International Conference on Robotics and Automation*, pp 331-337.

69. T-J.Pan and R.C.Luo, 1990, Motion planning for mobile robots in a dynamic environment with moving obstacles., *IEEE International Conference on Robotics and Automation*, pp 578-583.

70. H.Noborio, 1990, Several path-planning algorithms of a mobile robot for an uncertain workspace and their evaluation., *Proceedings of the IEEE International Workshop on Intelligent Motion Control*, Vol. 1, pp 289-294.

71. H.Noborio and J.Hashime, 1991, A feasible collision-free and deadlock-free path-planning algorithm in a certain workspace where multiple robots move flexibly., *IEEE International Workshop on Intelligent Robots and Systems IROS*, pp 1074-1079.

72. J.Borenstein and Y.Koren, 1990, Real-time obstacle avoidance for fast mobile robots in cluttered environments., *IEEE International Conference on Robotics and Automation*, pp 572-577.

73. M.Bennamoun, A.A.Masoud, M.A.Ramsay and M.M.Bayoumi, 1991, Avoidance of unknown obstacles using proximity fields., *IEEE International Workshop on Intelligent Robots and Systems IROS*, pp 435-440.

74. H.Noborio, 1992, A collision-free and deadlock-free path-planning algorithm for multiple mobile robots without mutual communication., *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, pp 479-486.

75. T.Wang, N.E Gough and Q.A Mehdi, 1997, Path planning for AGVs based on the environment topology and fuzzy reasoning., *Proceedings of the 12th International conference on Systems Engineering*, pp 729-734.

76. Y.Davidor ,1993, Analogous crossover., *Proceeding of 3rd International Conference on Genetic Algorithms*, pp 98-103.

77. T.Shibata and T.Fukuda, 1994, Coordination in evolutionary multi-agent systems using fuzzy and genetic algorithm., *Control Engineering Practice*, Vol. 2, No. 1, pp 103-111.

78. I.J.Griffiths, T.Wang, N.E.Gough and Q.H.Mehdi, 1997, Scheduling AGV tasks by a genetic algorithm., *1st Workshop on Recent Advances in Mobile Robots*, pp 48-57.

79. Z.Michalewicz, 1994,Genetic algorithms + data structures = evolution programs., *2nd Edition, Springer-Verlag*, pp 261-269.

80. E.Mazer, J.M.Ahuactzin, E-G Talbi and P.Bessiere, 1994, The ariadne's clew algorithm., *International Conference on Simulation of Adaptive Behavior 2*, pp 182-188.

81. M.O.Berger, O.Kubitz and T.Sperlbaum-Weigt, 1995, Intra- and inter-robot communication., *Proceedings of 3rd International Symposium on Intelligent Robotic Systems*, pp 105-112.

82. R.A.Brooks, 1991, Intelligence without reason., *Proceedings of the 12th International Conference on Artificial Intelligence*, vol. 1, pp 569-595.

83. L.Steels, 1994, The artificial life roots of artificial intelligence., *Artificial Life Journal*, vol. 1, pp 75-110.

84. F.M.J Verschure, B.J.A Kröse and R.Pfeifer, 1992, Distributed adaptive control: The self-organisation of structured behavior., *Robotics and Autonomous Systems 9*, pp 181-196.

85. K.Jin, P.Liang and G.Beni, 1994, Stability of synchronized distributed control of discrete swarm structures. *IEEE International Conference on Robotics and Automation*, pp 1033-1038.

86. T.Fukuda and S.Nakagawa, 1987, A dynamically reconfigurable robotic system (concept of a system and optimal configurations)., *International Conference on Industrial Electronics Control and Instrumentation*, pp 588-595.

87. H.Asama, A.Matsumoto and Y.Ishida, 1989, Design of an autonomous and distributed robot system: ACTRESS, *International Workshop on Intelligent Robotic Systems*, pp 283-290.

88. P.Caloud, W.Choi, J-C Latombe, Le C. Pape and M.Yin, 1990, Indoor automation with many robots., *International Workshop on Intelligent Robotic Systems*, pp 67-72.

89. D.McFarland, 1994, Towards robot cooperation., *Proceedings of the simulation of adaptive behaviour conference*, pp 440-444.

90. Y-U Cao, A.S.Fukunaga and A.B.Kahng, 1997, Cooperative mobile robotics: Antecedants and directions., *Autonomous Robots 4*, pp 7-27.

91. G.Dudek, M.R.M Jenkin, E.Milios and B.Wilkes, 1996, A Taxonomy for multi-agent robotics., *Autonomous Robots 3*, pp 375-397.

92. R.Arkin, 1992, Cooperation without communication: multi agent schema-based robot navigation., *Journal of Robotic Systems*, pp 351-364.

93. L.Steels, 1990, Cooperation between distributed agents through self-organisation., *In European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pp 175-195.

94. R.Beckers, O.E.Holland and J.L.Deneubourg, 1994, From local actions to global tasks: Stigmergy and collective robotics., *Artificial Life IV*, pp 181-189.

95. S.Sen, M.Sekaran and J.Hale, 1994, Learning to coordinate without sharing information., *Proceedings of AAAI*, pp 426-431.

96. Y.Toquenaga, I.Kajitani and T.Hoshino, 1994, Egrets of a feather flock together., *Artificial Life IV*, pp 140-151.

97. J.K.Hodgins and D.C.Borgan, 1994, Robot herds: Group behaviors for systems with significant dynamics., *Artificial Life IV*, pp 319-324.

98. D.C.Brogan and J.K.Hodgins, 1997, Group behaviors for systems with significant dynamics., *Autonomous Robots 4*, pp 137-153.

99. C.R.Kube and H.Zang, 1997, Task modelling in collective robotics., *Autonomous Robots 4*, pp 53-72.

100. M.J.Mataric, 1992, Designing emergent behaviors: From local interactions to collective intelligence., *Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour*, MIT press, pp 432-441.

101. T.Ueyama, T.Fukuda and F.Arai, 1992, Configuration of communication structure for distributed robot systems., *IEEE International Conference on Robotics and Automation*, pp 807-812.

102. J.Wang, 1994, On sign-board based inter-robot communication in distributed robotic systems., *IEEE International Conference on Robotics and Automation*, pp 1045-1050.

103. F-C.Lin and J.Y-J.Hsu, 1997, Cooperation protocols in multi-agent robotic systems., *Autonomous Robots 4*, pp 175-198.

104. H.Asama, K.Ozaki, H.Itakura, A.Matsumoto, Y.Ishida and I.Endo, 1991, Collision avoidance among multiple mobile robots based on rules and communication., *International Workshop on Intelligent Robotic Systems*, pp 1215-1220.

105. T.Fukuda and K.Sekiyama, 1994, Communication reduction with risk estimate for multiple robotic systems., *IEEE International Conference on Robotics and Automation*, pp 2864-2869.

106. L.E.Parker, 1995, The effect of action recognition and robot awareness in cooperative robotic teams., *International Workshop on Intelligent Robotic Systems*, pp 212-219.

107. E.Yoshida, T.Arai, J.Ota and T.Miki, 1994, Effect of grouping in local communication systems of multiple mobile robots., *International Workshop on Intelligent Robotic Systems*, pp 808-815.

108. T.Ueyama and T.Fukuda, 1993, Knowledge acquisition and distributed decision making -cellular robotics approach using genetic algorithms based on local knowledge and local communication., *IEEE Conference on Robotics and Automation*, pp 167-172.

109. R.C.Arkin and J.D.Hobbs, 1992, Dimensions of communication and social organisation in multi-agent robotic systems., *Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour*, pp 486-493.

110. R.C.Arkin, T.Balch and E.Nitz, 1993, Communication of behavioural state in multi-agent retrieval tasks. *Proceedings of the IEEE International Conference on Robotics and Automation*, pp 588-594.

111. H.Yanco and L.Stein, 1993, An adaptive communication protocol for cooperating mobile robots., *From animals to animats 2*, MIT press, pp 478-485.

112. M.Pearce, R.Arkin and A.Ram, 1992, The learning of reactive control parameters through genetic algorithms., *IEEE International conference on intelligent robots and systems*, pp 65-70.

113. A.Ram, R.Arkin, G.Boone and M.Pearce, 1994, Using genetic algorithms to learn reactive control parameters for autonomous robotic navigation., *Adaptive behaviour, 2(3)*, pp 277-304.

114. F.Menczer and R.K.Belew, 1994, Evolving sensors in environments of controlled complexity., *Artificial Life IV*, pp 210-221.

115. M.Colombetti and M.Dorigo, 1992, Learning to control an autonomous robot by distributed genetic algorithms., *Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour*, MIT press, pp 305-312.

116. O.Miglino, K.Nafasi, and C.E.Taylor, 1996, Selection for wandering behaviour in a small robot, *Artificial life*, "http://www.cogs.susx.ac.uk/users/ezequiel/alife-page/evorob.html".

117. H.H.Lund, 1995, Evolving robot control systems, *Proceedings of the 1NWGA*, University of Vaasa, "http://www.cogs.susx.ac.uk/users/ezequiel/alife-page/evorob.html".

118. D.Floreano and F.Mondada, 1996, Evolution of homing navigation in a real mobile robot., *IEEE Transactions on Systems Man and Cybernetics*, vol. 26, No. 3, pp 396-407.

119. B.J.MacLennan, 1990, Evolution of communication in a population of simple machines., *Technical report*, Computer science department CS-90-104, University of Tennessee.

120. B.MacLennan, 1991, Synthetic ethology: An approach to the study of communication., *Artificial Life II*, Addison-Wesley, pp 631-657.

121. G.M.Werner and M.G.Dyer, 1990, Evolution of communication in artificial organisms, *Technical report*, UCLA-AI-90-06.

122. G.M.Werner and M.G.Dyer, 1991, Evolution of communication in artificial organisms., *Artificial Life II*, Addison-Wesley, pp 659-687.

123. J.Noble and D.Cliff, 1996, On simulating the evolution of communication, *Proceedings of the 4th International conference on simulation and adaptive behaviour*, MIT press, "http://www.cogs.susx.ac.uk/users/ezequiel/alife-page/behavior.html".

124. C.W.Reynolds, 1994, Competition, coevolution and the game of tag., *Artificial Life IV*, pp 60-69.

125. A.Qureshi, 1996, Evolving agents., *Genetic programming 96 proceedings of the 1st annual conference*, MIT press, pp 369-374.

126. M.Oliphant, 1996, The dilemma of saussurean communication., *BioSystems*, 37(1-2), pp 31-38.

127. D.Andre, 1994, Evolution of mapmaking ability: strategies for the evolution of learning, planning and memory using genetic programming., *Proceedings IEEE world congress on computational intelligence*, Volume 1, IEEE press, pp 250-255.

128. S.G.Handley, 1994, The automatic generations of plans for a mobile robot via genetic programming with automatically defined functions., *Advances in genetic programming, chapter 18*, MIT press, pp 391-407.

129. M.Crosbie and E.H.Spafford, 1995, Applying genetic programming to intrusion detection., *AAAI fall symposium on genetic programming*, pp 1-8.

130. M.Crosbie and E.H.Spafford, 1996, Evolving event-driven programs., *Genetic programming '96 Proceedings of the 1st conference*, MIT press, pp 273-278.

131. G.F.Spencer, 1994., Automatic generation of programs for crawling and walking., *Advances in genetic programming, chapter 15*, MIT press, pp 335-353.

132. K.Sims, 1987, Flocks, herds and schools: A distributed behavioural model., *SIGGRAPH '87*, 21(4), pp 25-34.

133. K.Sims, 1994, Evolving 3D morphology and behaviour by competition., *Artificial Life IV*, MIT press, pp 28-39.

134. K.Sims, 1994, Evolving virtual creatures., *Computer graphics annual conference series*, pp 15-22.

135. C.W.Reynolds, 1992, An evolved, vision based behavioural model of coordinated group motion., *From animals to animats the Proceedings of the Simulation of Adaptive Behaviour Conference*, MIT press, pp 384-393.

136. B.J.MacLennan and G.M.Burghardt, 1994, Synthetic ethology and the evolution of cooperative communication., Adaptive behaviour 2(2), pp 161-188.

137. T.Haynes, 1995, Evolving multiagent coordination strategies with genetic programming, *Technical report*, UTULSA-MCS-95-04, University of Tulsa.

138. T.Haynes, R.Wainwright and S Sen, 1995, Evolving cooperative strategies, *Proceedings of the 1st International Conference on Multiagent Systems*, pp 450-455.

139. T.Haynes, S.Sen, D.Schoenefeld and R.Wainwright, 1995, Evolving a team., *AAAI Fall symposium on genetic programming*, pp 23-30.

140. T.Haynes, 1995, Evolving behavioral strategies in predators and prey., *Workshop on Adaptation and Learning in Multiagent Systems*, pp 32-37.

141. T.Haynes, 1994, A simulation of adaptive agents in a mobile environment., *MSc Thesis*, Dept. of Mathematical and computer sciences, University of Tulsa, Tulsa.

142. E.A Di Paolo, 1996, Investigation into the evolution of communication behaviors., *Research paper* 445 University of Sussex.

143. E.A Di Paolo, 1997, Social coordination and spatial organization: Steps towards the evolution of communication., *The 4th European Conference on Artificial Life*, "http://www.cogs.susx.ac.uk/ecal97/present.html/".

144. P. de Bourcier and M.Wheeler, 1997, The truth is out there: the evolution of reliability in aggressive communication systems., *The 4th European Conference on Artificial Life ECAL*, "http://www.cogs.susx.ac.uk/ecal97/present.html/".

145. C.Fyfe and D.Livingstone, 1997, Developing a community language., *The 4th European Conference on Artificial Life*, "http://www.cogs.susx.ac.uk/ecal97/present.html/".

146. A.Billard and K.Dautenhahn, 1997, The social aspect of communication: a case study in the use and usefulness of communication for embodied agents., *The 4th European Conference on Artificial Life*, "http://www.cogs.susx.ac.uk/ecal97/present.html/".

147. L.Steels, 1996, The spontaneous self-organization of an adaptive language., *Machine Intelligence 15*, Oxford university press, "http://www.cogs.susx.ac.uk/users/ezequiel/alife-page/behavior.html/".

148. L.Steels, 1996, A self-organizing spatial vocabulary., *Artificial Life Journal*, 3(2), pp 319-326.

149. L.Steels, 1996, Emergent adaptive lexicons., *From animals to animats Proceedings of the Simulation of Adaptive Behaviour Conference*, MIT press, http://www.cogs.susx.ac.uk/users/ezequiel/alife-page/behavior.html.

150. Y.Maeda, T.Sasaki and M.Tokoro, 1997, Self re-organizing of language triggered by "language contact"., *The 4th European Conference on Artificial Life*, "http://www.cogs.susx.ac.uk/ecal97/present.html/".

# Appendix A

# Miscellaneous

## A.1 DRONE PROGRAMS

The two system blocks below (Figures A.1 and A.2) contain the code used by the drones in part I of the testing approach used in task 2. The first system block contains code that gives rise to a smooth and relatively predictable set of movements for the drones. The next system block contains code used to produce jittery and unpredictable motion in the drones. Each of the system blocks also has a simple collision resolution strategies.

```
system[
    main{0,-1}0[
          (+
               (Bk
                    (/ (NoMv)   0.15720)
                )
               (PA
                    (+
                         (*   0.46440 r?)
                         (NoMv)
                     )
                    (+
                         (*   0.31400 r?)
                         (PA   0.65600   0.50000)
                     )
                )
          )
      ]_Main
 ]_Sys
```

Figure A.1 Code for smooth moving drone.

```
system[
    main{0,-1}0[
          (+
               (Bk
                    (+ (NoMv)  (NoMv))
                )
               (*
                    (NoMv)
                    (* r?   0.13000)
                )
          )
      ]_Main
 ]_Sys
```

Figure A.2 Code for unpredictably moving drone.

## A.2 TASK II FITNESS EVALUATION CODE:

The evaluation of the fitness of a program used in task II (chapter 6) is a three-stage process, firstly the code in figure A.3 is used to evaluate the ongoing performance of the robots each time a program is swapped out by the interpreter. Secondly, at the end of each test the code shown in figure A.4 is used to calculate the average performance for the test. Finally the code in figure A.5 is used to generate the overall fitness of the program across all tests and with the addition of any penalties.

```
if (drone_test)
 { // part I of testing currently being used
  i=0;
  for (j=Rn;j<Rn+NumOfDrones;j++)
    {
     d=0.0;
     if (i!=j)
      {  // calculate distance apart
       d=√((x_i-x_j)²+(y_i-y_j)²);

        // check to see if in safety or danger zone
       if (d<=(r_i+r_j)*1.9)
         if (d > (r_i+r_j)*1.1)
           d=0.0;
         else
             d=d*5.0;
       }

       // add current separation to value so far
      S=S +(d*2.0);
      }
  }
else // part II of testing currently being used
    for (i=0;i<Rn;i++)
     for (j=0;j<Rn;j++)
       {
        d=0.0;
        if (i!=j)
         {  // calculate distance apart
          d=√((x_i-x_j)²+(y_i-y_j)²);

           // check to see if in safety or danger zone
          if (d<=(r_i+r_j)*1.9)
            if (d >(r_i+r_j)*1.1)
              d=0.0;
            else
                d=d*5.0;
          }

           // add current separation to value so far
         S=S+d;
         }
C=C+1;  // count the number of summations made
```

Figure A.3 Algorithm to calculate on going performance of robots.

In Figure A.3: Rn is the maximum number of robots which can run the test program, $(x_i,y_i)$ are the co-ordinates of robot **i**, $(x_j,y_j)$ are the co-ordinates of robot **j**, $\mathbf{r_i}$ is the radius of robot **i**, **S** is the sum of the displacements between robots and **C** is the number of times the summations were made.

```
for (i=0;i<Rn;i++)
  av[0][run]+=(100.0*(Vi/Cyi));
av[0][run]= av[0][run]/Rn;
av[1][run]= S/C;
```

Figure A.4 Algorithm to calculate average fitness of program for a test.

In Figure A.4: **Rn** is the maximum number of robots which can run in the test program, $\mathbf{Cy_i}$ is a variable which counts the number of interpreter cycles robot **i** ran for during a test (the counter stops when the robot runs out of energy), $\mathbf{V_i}$ is a variable which is used to count the number of collisions robot **i** was involved in during the test (the counting stops when the robot runs out of energy), run indicates the number of tests performed so far for the current program, **av** is a multi-dimensional array holding the average performance of the program in each of the tests, **S** is the sum of the displacements between robots and **C** is the number of times the summations were made.

```
d1=0.0;
d2=0.0;
mx1=0;
mx2=0;
for (i=0;i<4;i++)
 {
  if (av[0][i]>mx1)
    mx1=av[0][i];          // find max average collision percentage
  if (av[1][i]>mx2)
    mx2=av[1][i];          // find max average distance apart
  d1=d1+(av[0][i]/ 4);     // average collision percentage over all tests
  d2=d2+(av[1][i]/ 4);     // average distance apart over all tests
  }
if (d1==0.0)
  d1=100.0;
else
    d1=d1+mx1;
d2=d2+mx2;
f0= d2+((d1/2.0)*(1+(Rn-2)));  // set constraint 1
f1= d1;                        // set constraint 2
```

Figure A.5 Algorithm to calculate final fitness of program across all test runs.

Some of the some what longer controllers produce as a result of the experiments employing this fitness evaluation method (task II, chapter 6) are presented below, and are proceeded by a table indicating the nature of the controller, the experiment if was produced in and what its fitness value was.

| APPENDIX FIGURE NUMBER | EXPERIMENT PRODUCED IN | SUMMARY OF CONTROLLER | FITNESS USING COMMUNICA -TION | | FITNESS USING NO COMMUNICA -TION | |
|---|---|---|---|---|---|---|
| | | | constraint 1 | constraint 2 | constraint 1 | constraint 2 |
| A.6 | I | Multi-type controller dependent on range *(Gdx; Gdy).* | 459 | 73 | 474 | 84 |
| A.7 | II | Beneficial use of communicated information but usage of it unclear *(Gdx; Gdy).* | 435 | 38 | 551 | 38 |
| A.8 | II | Beneficial but unclear use of communicated information *(Gdx; Gdy).* | **391** | **20** | **395** | **34** |
| A.9 | III | Poor overall performance but dependent on communicated information for performance *(Gdx; Gdy -global-).* | 684 | 32 | 709 | 40 |
| A.10 | III | Poor overall performance but heavily dependent on communicated information for performance *(Gdx;Gdy -global-).* | 686 | 35 | 1201 | 130 |
| A.11 | III | Same as **Error! Reference source not found.** *(Gdx; Gdy - global-).* | 534 | 12 | 1140 | 117 |

Table A.1 List of some of the long controllers produced in experiments carried out in chapter 6.

```
system[
    main{0,-1}12[
        (IFGE
            (OnLSC
                (/
                    (-
                        (/ r? (Gdx))
                        (OnLSL
                            (Max
                                (IFGE  0.32608  0.06667
                                   THEN (_)
                                   ELSE
                                    (RInv (- (OnLSC r? (Gdy))  (OnLSL r?  0.68800)))
                                )
                                (PA
                                    (-
                                        (- (_) (_))
                                        (OnLSL
                                            (OnLSR
                                                (+
                                                    (SH (NoMv))
                                                    (OnLSL r? r?)
                                                )
                                                (*
                                                    (- (Gdy) (Bk (NoMv)))
                                                    (Gdy)
                                                )
                                            )
                                            (_)
                                        )
                                    )
                                    (IFGE (NoMv)  0.84088
                                       THEN (Bk (Gdx))
                                       ELSE  0.18973
                                    )
                                )
                            )
                            (_)
                        )
                    )
                    (OnLSL (NoMv) (NoMv))
                )
                (Max (Gdx) r?)
            )
            (*
                r?
                (Min (- (Bk (NoMv)) (SH r?)) (PA (_) r?) )
            )
              THEN
                (OnLSC
                    (*
                        (Gdy)
                        (OnLSR
                            (+
                                (-
                                    (RInv (Gdx))
                                    (RInv (RInv  0.53749))
                                )
                                (/ (_) (Gdy))
                            )
                            (PA (NoMv)  0.81908)
                        )
                    )
                    0.46552
                )
              ELSE
                (OnLSR (_) (NoMv))
        )
    ]_Main
]_Sys
```

Figure A.6 Range dependent controller type featuring optimal usage of communicated information,

```
system[
    main{0,-1}12[
        (+
            (OnLSC
                (/
                    (PA
                        (-
                            (/  0.20591 (Gdx))
                            (*
                                (+
                                    (IFGE
                                        (RInv
                                            (/
                                                (Flip (Gdx))
                                                (/
                                                    (OnLSR
                                                        (OnLSL
                                                            (Bk
                                                                (OnLSR
                                                                    (+
                                                                      (OnLSL (_) (Gdy))
                                                                      (NoMv)
                                                                    )
                                                                    (- (NoMv) (NoMv))
                                                                )
                                                            )
                                                            0.81921
                                                        )
                                                    (_)
                                                    )
                                                (Gdy)
                                                )
                                            )
                                        )
                                    (Min
                                        (* (_) (Min r? (Gdx)))
                                        (Bk (Min (Gdx) (NoMv)))
                                    )
                                      THEN (NoMv)
                                      ELSE  0.01418
                                )
                                (OnLSR
                                    0.80232
                                    (/ (OnLSR (Bk r?) (_)) (Gdx))
                                )
                            )
                            (_)
                        )
                    )
                    (+
                        (Max
                            (IFGE
                                (- (Gdx) (Flip (NoMv)))
                                (/ (* (NoMv) (NoMv)) (Gdx))
                                THEN
                                 (+
                                     (+ (RInv (NoMv)) (PA  0.11305 (Gdy)))
                                     (* (OnLSC (_) (Gdy)) (_))
                                 )
                                ELSE
                                 (+
                                     (/ r? (_))
                                     (SH
                                         (+
                                             (- (Gdx) r?)
                                             (*
                                                 (-
                                                     (OnLSC (_) (Gdx))
                                                     (- (Gdx) (Gdy))
                                                 )
                                                 (* r? (Bk (NoMv)))
                                             )
                                         )
                                     )
                                 )
                            )
                            0.33964
                        )
```

```
                        (- r? (NoMv))
                    )
                )
                (Bk (Gdx))
            )
            (Bk
                (Bk
                    (OnLSR
                        (OnLSC
                            (Min (_) (Min (_) (NoMv)))
                            (-
                                (+
                                    (OnLSR
                                        (-
                                            (OnLSR
                                                (*
                                                    (OnLSC
                                                        (+
                                                            (-  0.13462 (Gdx))
                                                            (Min  0.71134 r?)
                                                        )
                                                        (Bk  0.72876)
                                                    )
                                                    (NoMv)
                                                )
                                                (Gdx)
                                            )
                                            (NoMv)
                                        )
                                        (+ (RInv (Gdy)) 0.59914)
                                    )
                                    (OnLSR
                                        (Bk (/ (RInv (Gdy)) (Gdx)))
                                        (_)
                                    )
                                )
                                (-
                                    (Flip (Gdy))
                                    (IFGE r?  0.95315
                                       THEN (NoMv)
                                       ELSE r?
                                    )
                                )
                            )
                            0.84064
                        )
                    )
                )
            )
            0.57045
        )
    ]_Main
]_Sys
```

Figure A.7 Controller exhibiting beneficial yet unclear usage of communicated information.

```
system[
    main{0,-1}12[
        (+
            (IFGE
                0.77843
                (OnLSR
                    (RInv
                        (Min
                            (* (_) (Flip  0.62932))
                            (Flip
                                (-
                                    (IFGE (NoMv) (_)
                                       THEN (_)
                                       ELSE (Gdx)
                                    )
                                    (Bk
                                        (/
                                            (-
                                                (IFGE  0.65315  0.05218
                                                   THEN (Gdx)
                                                   ELSE (NoMv)
                                                )
                                                (Flip
                                                    (OnLSR
                                                        (/
                                                            (*
                                                                (Gdy)
                                                                (OnLSL
                                                                    (_)
                                                                    (Max
                                                                        (NoMv)
                                                                        (/
                                                                            (* (OnLSC
                                                                                0.01944
                                                                                r?
                                                                                )
                                                                                (NoMv)
                                                                            )
                                                                            (/ (Bk
                                                                                (Gdx)
                                                                               )
                                                                               (/
                                                                                0.98241
                                                                                (_)
                                                                                )
                                                                            )
                                                                        )
                                                                    )
                                                                )
                                                            )
                                                            0.49280
                                                        )
                                                        r?
                                                    )
                                                )
                                            )
                                            (/
                                                (* (Gdx) (Bk (SH  0.81234)))
                                                (Gdy)
                                            )
                                        )
                                    )
                                )
                            )
                        )
                    )
                    (Bk (Gdy))
                )
                  THEN (Bk  0.68067)
                  ELSE (* (OnLSC  0.01208 (_)) 0.60566)
            )
            0.61048
        )
    ]_Main
]_Sys
```

Figure A.8 Another controller exhibiting beneficial yet unclear usage of communicated information.

```
system[
    main{0,-1}7[
        (+
            0.69194
            (IFGE
                (+
                    (Max (NoMv) (_))
                    (-
                        (Min (Flip (Gdy)) (Gdy))
                        (SH
                            (-
                                (Bk (IFGE (Gdx)  0.54730 THEN r? ELSE r?))
                                (RInv
                                    (+  (Bk (Max
                                                (-
                                                    (* (SH (Gdx)) (Min  0.37084 (Gdy)))
                                                    (*
                                                        (RInv (Gdx))
                                                        (IFGE
                                                            (/
                                                               (+
                                                                  (Bk  0.10228)
                                                                  (RInv (Gdx))
                                                               )
                                                               0.12821
                                                            )
                                                            (RInv (Flip (_)))
                                                               THEN (*
                                                                        (Bk (Gdy))
                                                                        (Bk (Gdx))
                                                                    )
                                                               ELSE (Gdy)
                                                        )
                                                    )
                                                )
                                            )
                                        (+ (RInv
                                                (Min
                                                    (Max (_)  0.60746)
                                                    (Max
                                                        (Null
                                                            (/
                                                              (Min (Gdy) r?)
                                                              (Min
                                                                 0.88618
                                                                 (NoMv)
                                                              )
                                                            )
                                                        )
                                                        (NoMv)
                                                    )
                                                )
                                            )
                                        (SH (+ (Bk (NoMv)) (Null r?)))
                                    )
                                )
                            )
                            (RInv (+ (SH  0.31136) (SH  0.14135)))
                        )
                    )
                )
                (+  (PA (_) (_)) (SH (Max (_) (NoMv))))
                   THEN (Null (NoMv))
                   ELSE
                    (/  r?
                        (Bk
                            (Flip
                                (/
                                    (* (Gdy) (_))
                                    (Flip
                                        (Max
                                            (IFGE (Gdx) (PA (_) (NoMv))
                                               THEN  (/ (Null  0.68857) (Gdx))
                                               ELSE  (+ (RInv (_)) (NoMv))
                                            )
                                            (* (Bk (NoMv)) (PA (Gdx) (RInv r?)))
                                        )
```

```
                                        )
                                    )
                                )
                            )
                        )
                    )
                )
            ]_Main
        ]_Sys
```

Figure A.9 Poor controller, however still dependent on communicated information for performance.

```
system[
    main{0,-1}7[
        (/  (Max
                (Bk (Null (Min (NoMv) r?)))
                (IFGE
                    r?
                    (Flip
                        (Max
                            (Null (- (RInv  0.28678) (Flip (/ (Bk (Gdx)) (Bk (Gdy))))))
                            (+  (/  (+  0.20891
                                        (+  (NoMv)
                                            (RInv
                                                (/  (NoMv)
                                                    (+
                                                        (+  (SH (Bk (Gdx)))
                                                            (IFGE
                                                                (Min
                                                                    (-  (/ (Gdy) (_))
                                                                        (IFGE
                                                                          (Bk  0.04314)
                                                                          (/
                                                                             (IFGE
                                                                                 r?
                                                                                 (Gdy)
                                                                               THEN
                                                                                   (_)
                                                                               ELSE
                                                                                    r?
                                                                             )
                                                                          (Gdx)
                                                                        )
                                                                        THEN
                                                                        (-
                                                                          (PA
                                                                             0.41805
                                                                              r?
                                                                          )
                                                                          (IFGE
                                                                             (NoMv)
                                                                             0.28155
                                                                             THEN r?
                                                                             ELSE r?
                                                                          )
                                                                        )
                                                                        ELSE
                                                                        (*
                                                                          (Null r?)
                                                                          (-
                                                                            (Gdy)
                                                                            (NoMv)
                                                                          )
                                                                        )
                                                                    )
                                                                )
                                                            )
                                                        (-  (Bk (/  (-
                                                                      r?
                                                                      (*
                                                                        (/
                                                                          (-
                                                                        (Gdy)
                                                                        (NoMv)
                                                                            )
```

```
                                                
```

```
system[
    main{0,-1}7[
        (SH (Flip
                (+
                    (Bk (- (/ (Max r? r?) (Gdx)) (PA (Gdx) (NoMv))))
                    (*
                        (Gdy)
                        (-
                            (Max
                                (/ (Bk (Flip (RInv (Gdx)))) (Gdy))
                                (+ (Null (Gdx)) (/ 0.61209 (Bk (Gdx))))
                            )
                            (Max
                                (IFGE
                                    (Null
                                        (Bk
                                            (+
                                                (+
                                                    (SH (Bk  0.56701))
                                                    (Flip (* (NoMv) r?))
                                                )
                                                (Flip (NoMv))
                                            )
                                        )
                                    )
                                    (NoMv)
                                       THEN (NoMv)
                                       ELSE (_)
                                )
                                (Null (/ (Flip (NoMv)) (Bk (Gdx))))
                            )
                        )
                    )
                )
            )
        )
    ]_Main
]_Sys
```

Figure A.11 Best controller produced with globally transmitted displacement communication.

# APPENDIX_B

# PUBLICATIONS

I.Ashiru and C.A.Czarnecki, 1998, Experiments in evolving communication controllers for teams of mobile robots., *Accepted for publication in IEEE international conference on Robotics and Automation.*

I.Ashiru and C.A.Czarnecki, 1998, Evolving communication strategies for team of robot, *Accepted for publication in Journal of Microprocessor and Microsystems.*

I.Ashiru and C.A.Czarnecki, 1997, Designing controllers for multiple robot systems using genetic programming., *Proceedings of 42nd International Scientific Colloquium,* Technical University of Ilmenau, Ilmenau, Sept. 22-25, pp 355-360.

I.Ashiru and C.A.Czarnecki, 1997, Evolving communicating controllers for multiple robot systems., *12th International Conference on Systems Engineering*, ICSE '97, Coventry, Sept. 9-11, pp 33-38.

I.Ashiru and C.A.Czarnecki, 1997, Evolving Co-operative controllers for teams of robots., *Workshop on Recent Advances in Mobile Robots*, Leicester, July 1st, pp 58-66.

I.Ashiru, C.A.Czarnecki and T.W.Routen, 1996, Characteristics of a genetic based approach to path planning for mobile robots., *Journal of Network and Computer Applications*, Vol. 19, pp 149-169.

I.Ashiru, C.A.Czarnecki and T.W.Routen, 1995, Intelligent operators and optimal genetic based path planning for mobile robots., *International Conference on Recent Advances in Mechatronics*, Istanbul, August 14-16, pp 1018-1023.

I.Ashiru, C.A.Czarnecki and T.W.Routen, 1995, Genetic representations and mobile robot path planning, *Third International Symposium on Intelligent Robotic Systems*, Pisa, July 10-14, pp 175-184.

I.Ashiru, C.A.Czarnecki and T.W.Routen, 1995, Evolutionary motion planning for mobile robots., *ECML Workshop on Intelligent Robots*, Herallion, April 28-29, pp 1-6.

I.Ashiru and C.A.Czarnecki, 1995, Optimal motion planning for mobile robot path planning., *IEEE International Conference on Industrial Automation and Control*, Hyderabad.

BRITISH LIBRARY

IMAGING SERVICES NORTH
Boston Spa, Wetherby
West Yorkshire, LS23 7BQ
www.bl.uk

PAGES NOT SCANNED AT THE REQUEST OF THE UNIVERSITY

SEE ORIGINAL COPY OF THE THESIS FOR THIS MATERIAL

# Appendix C

# IMPLEMENTATION OF EVOLUTIONARY TOOLS

## C.1 INTRODUCTION

This appendix a brief description of the implementational details of both of the evolutionary methods developed and used in this thesis, along with a disk containing the code.

## C.2 GENETIC ALGORITHMS

The GA is implemented as a memory based program, in that all processing takes place within the internal memory of the computer. The system is designed to run on a single machine but the code is written such that it is portable across various platforms and compilers. A mixture of C and C++ is used in the coding of the GA tool. However, the majority of the code is written in C and C++ is to provide the class and inheritance framework required in implementing the GA. The GA tool is capable of both interactive (in which users can watch the development of populations as well as change many of the systems parameters and operators) and batch (where the system just processes the population, required for remote and over night running of the system) modes of running. In interactive mode, the status of the system can be saved at the end of each generation. This status information can then be used to re-start the system for where it left of in a number of ways (warm starts). The system accepts command line parameters to define initial operator configuration as well as which test case and scenarios to use. Associated with each run a number of files, these contain either a list of the best individuals found (history file), data and statistical information about each generation (best file) and a list of all individuals that have meet the tolerance requirements of the test cases (tolerance file). The inheritance structure of the code is given in Figure C.1.

In order to change the size of the population and other structures or properties of the GA, the appropriate global or inherited items must be changed and the system must then be re-compiled. The basic code for the GA can be found in directory a:\ga of the attached disk.

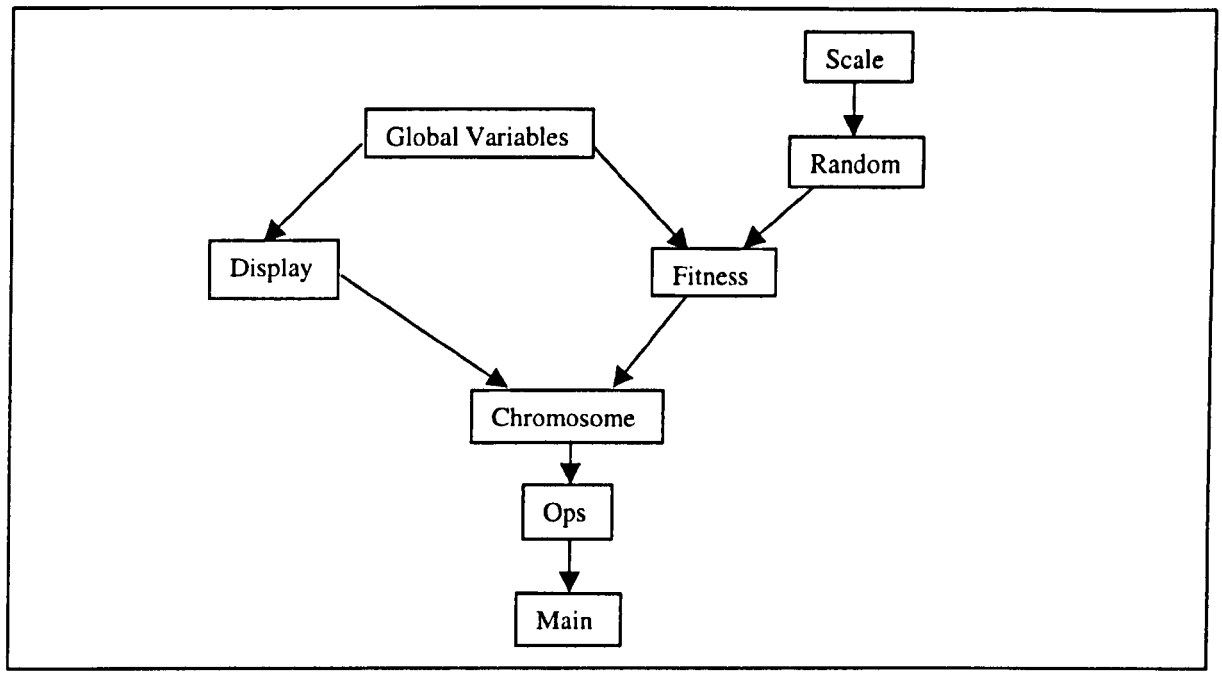



Figure C.1 Inheritance diagram for genetic algorithm implementation.

*Scale: This module contains routines for scaling data and for converting data such that its in a form that can be minimised, maximised or alter the distribution of a random function.*

*Random: This module contains data structures and routines to allow for multiple random streams; real, integer and bit random generators; biased distributions; random number generators without replacement and random sorts of finite amount of data.*

*Global variables: This is a class containing all the fully global variables and structures used by the GA.*

*Display: This class contains the graphics routines used by the GA. These routines are written using #define statements to enable portability.*

*Fitness: This class contains all functions and structures associated with the determining the fitness of a chromosome.*

*Chromosome: This class contains all the genetic operators for applying to the chromosome representation being manipulated by the GA.*

*Ops: This class is responsible for collating statistics on the run, generating and updating files associated with each run, cascading the interactive changes made to the system and co-ordinating the loading and saving of status information used for warm starts.*

*Main: This module is responsible for processing the command line parameters and for implementing the general GA algorithm flow diagram, in terms of function calls.*

## C.3 GENETIC PROGRAMMING

The GP is implemented as a disk based system, in that all the data to be manipulated is stored on the hard disk of the computer and only pulled into memory as needed. The system is designed to be run across distributed machines (i.e. distributed evaluation see section 5.5.3) and is written in a machine and compiler independent fashion. The language C is used for the bulk of the coding and its structural and abstract properties are derived through the use of C++ classes and inheritance mechanisms. The system is designed such that the GP engine is exchangeable. The system relies on a set of files to implement its software switchability (that is the altering of parameters or features of the system without having to re-compile the code), although changes to some elements of the system are non-software switchable. Files are also used to hold common code used by programs (see chapter 6 and 7). As in the GA implementation the GP can run in either interactive or batch mode, takes command line arguments, produces information files for each run and can have its current status saved and reloaded. The inheritance structure of the GP can be seen in Figure C.2. The code for the GP can be found in directory a:\gp of the disk attached.

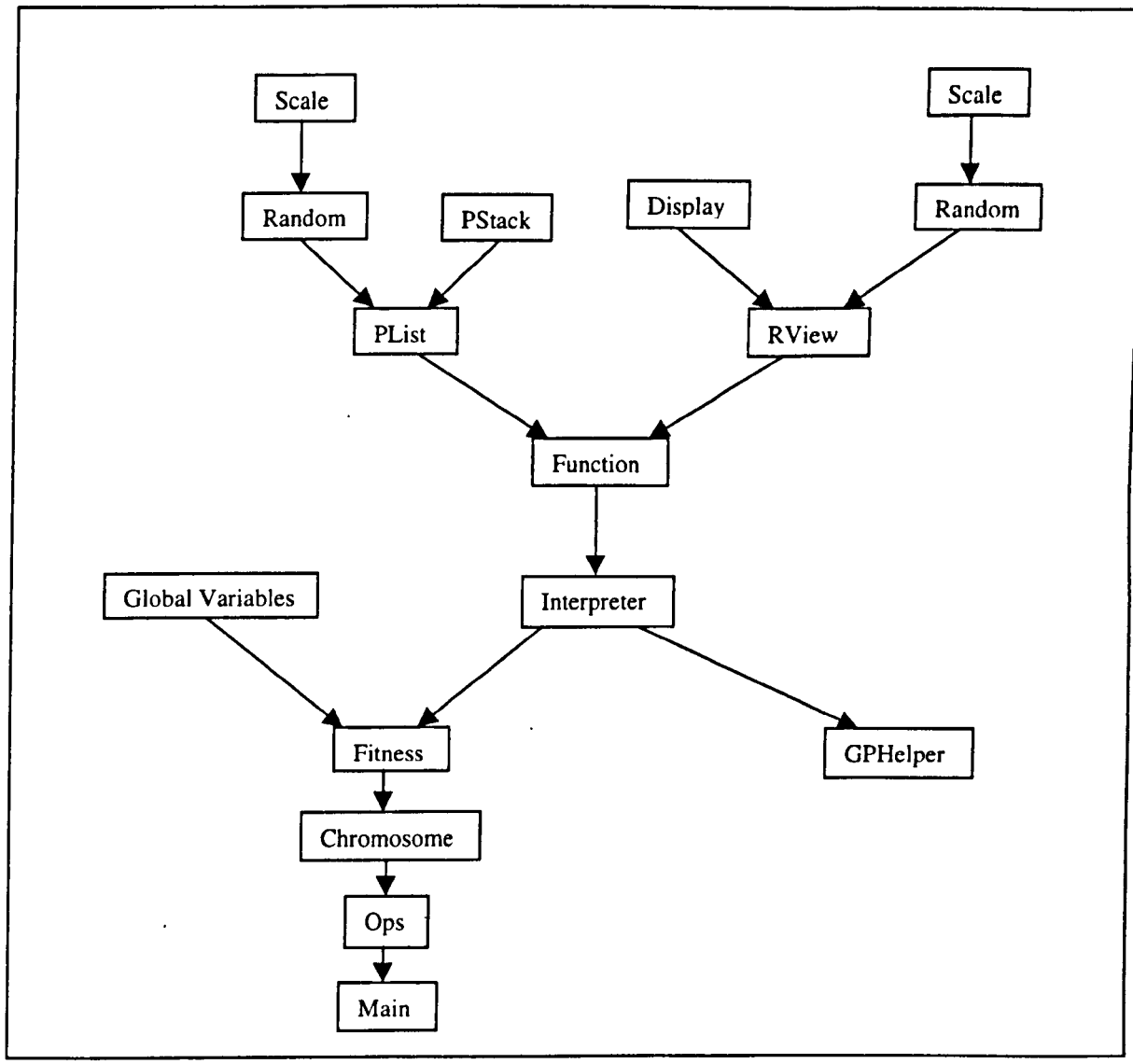



Figure C.2 Inheritance diagram for genetic programming implementation.

*Scale: This module contains routines for scaling data and for converting data such that its in a form that can be minimised, maximised or alter the distribution of a random function.*

*Random: This module contains data structures and routines to allow for multiple random streams; real, integer and bit random generators; biased distributions; random number generators without replacement and random sorts of finite amount of data.*

*PStack: This class provides stack functions used by the interpreter to execute programs, it also contains data structures associated with processes.*

*Display: This class contains the graphics routines used by the GP. These routines are written using #define statements to enable portability.*

*PList: This class contains the list data structures used to store programs as well as operators that allow the list to be processed, it also defines function/terminal sets and initialises the number of parameters associated with each function for input and output (the majority of these functions and those in PStack form the GP-engine).*

*RView: This class contains data structures and functions associated with the robot simulator.*

*Function: This class contains the initialisation process and the code for all the functions and terminals in built into the interpreter.*

*Interpreter: This class contains all the procedures required for the interpreter integrated with calls to the robot simulator.*

*Global variables: This is a class containing all the fully global variables and structures used by the GP.*

*Fitness: This class contains all functions and structures associated with the determining the fitness of a chromosome.*

*Chromosome: This class contains all the genetic operators for applying to the chromosome representation being manipulated by the GP (inherited from PList).*

*Ops: This class is responsible for collating statistics on each run, generating and*

*updating files associated with each run, cascading any interactive changes made to the system, co-ordinating the loading and saving of status information used for warm starts and also co-ordinating the creation and synchronisation of the distributed evaluation process.*

*Main: This module is responsible for processing the command line parameters and for implementing the general GP algorithm flow diagram, in terms of function calls.*

*GPHelper: This module is used to enable other processes to aid the Main module in the evaluation of controllers.*